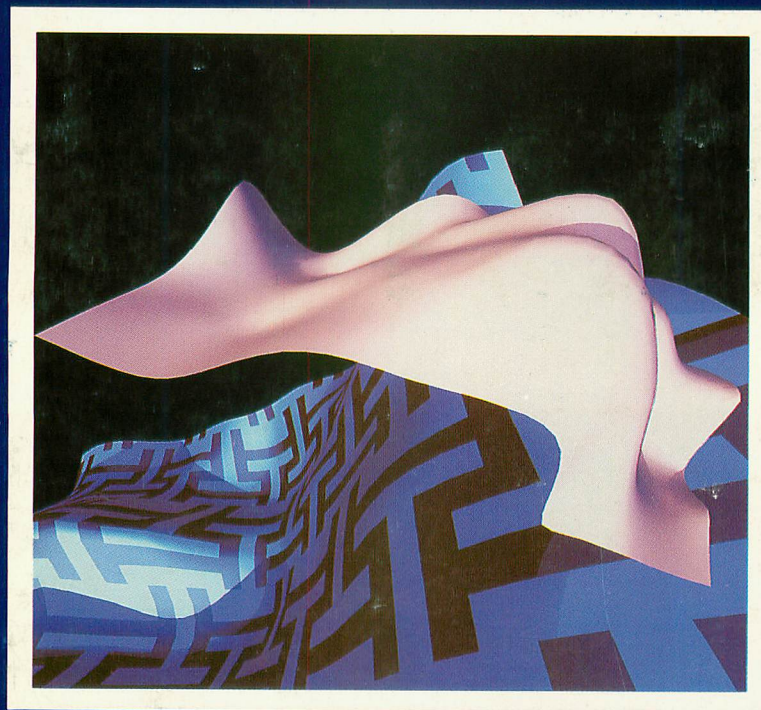


Proceedings

Graphics '86 Interface 86 Vision Interface '86

Comptes rendus



Canadian Information
Processing Society

Vancouver, B.C.
26-30 May/mai 1986



Canadian Man-Computer
Communications Society

Copyright 1986 by
Canadian Information Processing Society

Permission is granted to quote short excerpts and to reproduce figures and tables from these proceedings, provided that the source of such material is fully acknowledged.

ISSN 0713-5424

Published by the Canadian Information Processing Society

Conference sponsored by The Canadian Man-Computer Communications Society (CMCCS); Canadian Image Processing and Pattern Recognition Society (CIPPRS); Canadian Information Processing Society; in cooperation with the National Research Council of Canada

Membership information for the CMCCS, CIPPRS, as well as additional copies of this proceedings are available from:

Canadian Information Processing Society
243 College Street, 5th Floor
Toronto, Ontario
M5T 2Y1
tel.: (416) 593-4040

Printed in Canada

The Conference thanks the following organization for financial assistance

Apollo Computer Corporation
Vertigo Systems International

Front Cover

The picture represents the visual side of a relationship between musical and visual structures. The artist-composer is Th. Goldberg, who used the graphics language LIG developed by R. Ross at University of British Columbia. The digital image was recorded on film at MacDonald Dettwiler, Richmond, B.C. on the COLOR FIRE 240 film recorder, at a resolution of 8192 pixels per line.

Back Cover

Image created by Martin Dubetz and Judith McGillis at the Department of Computing Science, University of Alberta.

Graphics Interface '86

Copyright 1986 par
L'Association canadienne de l'informatique

Il est permis de citer de courts extraits et de reproduire des données ou tableaux du présent compte rendu, à condition d'en identifier clairement la source.

ISSN 0713-5424

Publié par l'Association canadienne de l'informatique

Conférence parrainé par la Société canadienne du dialogue homme-machine (SCDHM); l'Association canadienne de traitement d'images et reconnaissance des formes (ACTIRF); l'Association canadienne de l'informatique; en collaboration avec le Conseil national de recherches du Canada

Des renseignements sur la SCDHM, ACTIRF, et des exemplaires supplémentaires des comptes rendus sont disponibles à l'adresse suivante:

L'Association canadienne de traitement d'images et reconnaissance des formes
243, rue Collège, 5^e étage
Toronto (Ontario)
M5T 2Y1
tél.: (416) 593-4040

Imprime au Canada

Les organisateurs de la conférence expriment leur gratitude à l'organisme ci-après qui a financé l'évènement

Apollo Computer Canada Ltd.
Vertigo Systems International

La couverture

L'image représente le côté visuel de la relation entre les structures musicales et visuelles. L'artiste est Th. Goldberg qui a utilisé le langage graphique LIG développé par R. Ross à l'Université de Colombie-Britannique. L'image numérique a été enregistré sur film à MacDonald Dettwiler, Richmond (C.-B.) à l'aide d'un enregistreur de film COLOR FIRE 240 avec une résolution de 8192 pixels par ligne.

Le verso du livre

Une image créée par Martin Dubetz et Judith McGillis au département d'informatique, de l'Université d'Alberta.

Vision Interface '86

Proceedings / Comptes rendus

Graphics Interface '86
Vision Interface '86

26-30 May/mai
Vancouver, British Columbia

Message from the Conference General Chair

Graphics Interface '86 and Vision Interface '86 are another ground-breaking endeavour of the Canadian Man-Computer Communications Society. We are particularly proud of the fact that the event is the 12th Canadian conference devoted to computer graphics and the longest running computer graphics conference in North America.

This year we have accepted the Canadian Image Processing and Pattern Recognition Society as a partner. In the past, CIPPRS has participated in the program, but not as extensively nor as visibly. While the joint conference has certainly led to some coordination problems, it is hoped the synergism generated will be beneficial to all participants.

It is also significant that this is the first time that Graphics Interface is being held in Vancouver. We welcome the Vancouver participants who have not had an opportunity to attend GI Conferences in the past, and we hope for their continued support.

Putting together a program of this size and scope is a considerable task, and I want to thank the organizers for all their work. In particular I would like to mention Mark Green for putting together the Graphics Interface program, Morris Goldberg and Bob Woodham for the Vision Interface program, Gunther Schrack for local arrangements, and Marcell Wein for the Proceedings. Without volunteer efforts such as theirs, the conference would not be nearly as enjoyable.

Wayne A. Davis
President CMCCS and
Conference Chairman

Message from the Chairman of the Local Organizing Committee

A wide variety of events is taking place during Graphics Interface/Vision Interface '86. There are two days of tutorials to introduce new and interesting topics to those wanting more background information; three days of technical sessions presenting recent ideas, concepts and results in computer graphics and computer vision; a film show of the latest applications of technology to the world of entertainment; and a banquet and two receptions to allow the opportunity for the personal contacts and conversations so important at a professional conference.

Message du président de la conférence

Graphics Interface '86 et Vision Interface '86 constituent une autre activité novatrice de la Société canadienne du dialogue homme-machine. Nous sommes tout particulièrement fiers que cette douzième conférence canadienne consacrée à l'imagerie informatique soit la plus ancienne conférence dans le domaine en Amérique du Nord.

Cette année, l'Association canadienne de traitement d'images et reconnaissance des formes s'est jointe à nous pour l'organisation de la conférence. L'ACTIRF a participé au programme des conférences antérieures, mais jamais de manière aussi active ni aussi visible. Bien que le caractère conjoint de la conférence ait engendré certains problèmes de coordination, nous espérons que tous les participants bénéficieront des effets positifs de la "synergie" résultante.

De plus, il est important de souligner qu'il s'agit de la première fois que Graphics Interface a lieu à Vancouver. Nous souhaitons la bienvenue aux participants de Vancouver qui n'ont pas eu l'occasion d'assister aux conférences GI antérieures et nous espérons que leur participation se poursuivra à l'occasion des conférences futures.

L'établissement d'un programme de cette envergure est une tâche considérable et je tiens à remercier tous les organisateurs. Je veux mentionner en particulier Mark Green qui a coordonné le programme de Graphics Interface, Morris Goldberg et Bob Woodham qui ont coordonné le programme de Vision Interface, ainsi que Marcell Wein qui est responsable du compte rendu. Sans la participation bénévole de ces gens et d'autres personnes, le résultat final ne serait pas aussi satisfaisant.

Wayne A. Davis
Président de la SCDHM et
Président de la conférence

Message du Président du Comité d'organisation local

Le programme de Graphics Interface/Vision Interface '86 offre une gamme très variée d'activités. Tout d'abord, deux journées de séances d'étude portant sur des domaines neufs et captivants à l'intention des personnes qui désirent s'y familiariser; ensuite, trois jours d'ateliers techniques traitant d'idées, de concepts et de résultats récents dans les secteurs de l'imagerie informatique et de la vision informatique; un programme de films montrant certaines applications récentes de ces technologies au domaine des arts du spectacle; et finalement, un banquet et deux réceptions favorisant les conversations et contacts personnels, facette très importante d'une conférence professionnelle.

And there is more! Vancouver is an inviting city, nestled between the mountains and the sea with plentiful fine restaurants, entertainment, breathtaking sights, and fascinating neighbourhoods such as Gastown, Chinatown and Granville Island, to name but a few. Of course there is Expo '86, an entire summer of entertainment and information, concentrating on transportation and communication of the past, present and future. I hope you can find time to take advantage of at least some of these attractions.

A conference of this size depends entirely on the dedication of the volunteers who are organizing it. Much work has been invested, and I wish to express my sincere thanks to all my fellow committee members who gave much time and effort to make this conference possible. I hope you will benefit from it.

Gunther Schrack
University of British Columbia

Message from the Program Chair Graphics Interface '86

Over the years Graphics Interface has developed a reputation for a varied and interesting program. It is my hope that the 1986 program will meet the standards set by previous Graphics Interface conferences.

This year's program is a major departure from those of previous years, in that there are fewer parallel sessions. The Program Committee had to be more selective in the review of papers. I would like to thank the committee members for the extra effort required to review the papers (most of which were full papers, as opposed to the extended summaries submitted in previous years). The trend towards fewer parallel sessions should increase the quality of the papers presented at Graphics Interface.

Another major difference in 1986 is the inclusion of the Vision Interface conference. This new conference is being held in parallel with Graphics Interface, with sessions open to attendees of both conferences. Graphics Interface and Vision Interface will share several common sessions and this will encourage interaction between these two related fields.

Mark Green
University of Alberta

Et ce n'est pas tout! Vancouver est une ville très accueillante, sise entre la montagne et mer, et qui regorge d'excellents restaurants, de lieux de divertissement, de paysages à couper le souffle et de quartiers pittoresques comme, par exemple, Gastown, Chinatown et Granville Island. Sans parler bien sûr d'Expo '86, qui offrira pendant tout l'été un programme de divertissement et d'information, en particulier dans les domaines des transports et des télécommunications d'hier, d'aujourd'hui et de demain. Je vous souhaite d'avoir le temps de profiter d'au moins une partie de ces attractions.

Une conférence de cette envergure repose entièrement sur le travail des volontaires qui l'organisent. La somme de travail nécessaire a été énorme et je veux remercier tous les autres membres du comité d'organisation qui ont consacré beaucoup de temps et d'énergie à cette conférence; sans eux, le tenue de la conférence aurait été impossible. J'espère que la conférence vous sera profitable.

Gunther Schrack
University of British Columbia

Message du président du programme de Graphics Interface '86

Depuis ses débuts, Graphics Interface a acquis la réputation d'offrir un programme varié et intéressant. J'ose espérer que le programme de 1986 satisfera aux critères exigeants établis lors des conférences Graphics Interface précédentes.

Le programme de cette année se distingue nettement de celui des années précédentes puisqu'il comporte moins d'activités concomitantes. Le Comité du programme a dû étudier les communications proposées avec plus de rigueur. Je remercie les membres de ce comité pour le surcroît de travail qu'a exigé la lecture des textes (dont la plupart étaient des articles complets plutôt que des résumés détaillés comme les années précédentes). La diminution du nombre d'activités en parallèle devrait accroître la qualité d'ensemble des communications présentées lors de Graphics Interface.

L'autre nouvel aspect très important en 1986 est l'adjonction de la conférence Vision Interface. Cette nouvelle conférence se tient en parallèle à Graphics Interface et toutes les activités sont ouvertes aux membres des deux conférences. Plusieurs ateliers seront communs à Graphics Interface et Vision Interface, ce qui devrait favoriser les échanges entre ces deux domaines connexes.

Mark Green
University of Alberta

Message from the Program Co-chairs of Vision Interface

Welcome to Vision Interface '86, sponsored by the Canadian Image Processing and Pattern Recognition Society (CIPPRS). It is a great pleasure to begin a new series of vision conferences in conjunction with Graphics Interface '86, the 12th graphics conference sponsored by the Canadian Man-Computer Communications Society (CMCCS). Vision Interface '86 has relied heavily on the organizational work of the Graphics Interface '86 Planning Committee, for which we are most grateful. Hopefully, all attendees benefit from the joint tutorial program and technical sessions of the companion conferences. As Program Co-chairs of Vision Interface '86, we thank our invited speakers and all those who submitted papers. These proceedings become the permanent record of your efforts.

M. Goldberg
University of Ottawa

and

R.J. Woodham
University of British Columbia

Message des co-présidents du programme de Vision Interface

Nous vous souhaitons la bienvenue à Vision Interface '86, que commandite l'Association canadienne de traitement d'images et reconnaissance des formes (ACTIRF). C'est avec un vif plaisir que nous inaugurons une nouvelle série de conférences sur la visionique, en collaboration avec Graphics Interface '86, la douzième édition de cette conférence commanditée par la Société canadienne du dialogue homme-machine (SCDHM). Vision Interface '86 a beaucoup profité de l'expérience du comité d'organisation de Graphics Interface '86; nous lui exprimons toute notre reconnaissance. Nous espérons que tous les participants profiteront du programme commun de séances d'études et d'ateliers techniques des deux conférences. À titre de co-présidents de Vision Interface '86, nous remercions les conférenciers invités ainsi que tous ceux qui nous ont soumis des articles. Le présent compte rendu forme un témoignage permanent de vos travaux.

M. Goldberg
Université d'Ottawa

et

R.J. Woodham
University of British Columbia

ORGANIZING COMMITTEE / COMITÉ ORGANISATEUR

Conference Chairman/Président de la conférence	Wayne Davis, University of Alberta
GI '86 Program and Tutorials/Programme et cours d'instruction GI '86	Mark Green, University of Alberta
Program Committee/Comité du programme	W.W. Armstrong, University of Alberta T.W. Calvert, Simon Fraser University W.A. Davis, University of Alberta M. Dubetz, University of Alberta A. Fournier, University of Toronto M. Green, University of Alberta D.R. Hill, University of Calgary S. MacKay, NRCC/CNRC D. Peachey, University of Saskatchewan G.F. Schrack, University of British Columbia G. Singh, University of Alberta P.P. Tanner, University of Waterloo M. Tuori, Defence and Civil Institute of Environmental Medicine/Institut militaire et civil de médecine environnementale M. Wein, NRCC/CNRC
VI '86 Program/Programme VI '86	Morris Goldberg, University of Ottawa/Université d'Ottawa Robert Woodham, University of British Columbia
Proceedings Editors/Rédacteurs des comptes rendus	Marceli Wein, Evelyn M. Kidd, NRCC/CNRC
Treasurer/Trésorier	Mike Kendrick, Vertigo Systems International
Publicity/Publicité	Chander Khanna, Vertigo Systems International
Filmshow/Séance cinématographique	Severin Gaudet, Vertigo Systems International
Equipment Exhibition/Exposition de matériel	Tom Poiker, Simon Fraser University
Local Arrangements/Arrangements locaux	Gunther Schrack, University of British Columbia
Audio-Visuals/Audiovisuel	Norm Dadoun, University of British Columbia
Corporate Sponsorships/Parrainage collectif	John Dill, Microtel Pacific Research
Secretary/Secrétaire	Willie Laurilla, Capilano College
Member-at-Large/Membre sans titre particulier	Tom Calvert, Simon Fraser University

**CANADIAN MAN-COMPUTER COMMUNICATIONS SOCIETY /
LA SOCIÉTÉ CANADIENNE DU DIALOGUE HOMME-MACHINE**

President/Président

Dr. Wayne A. Davis
Department of Computing Science
University of Alberta
Edmonton, Alberta
T6G 2H1

Vice-President/Vice-président

Peter P. Tanner
Computer Graphics Laboratory
University of Waterloo
Waterloo, Ontario
N2L 3G1

Secretary-Treasurer/Secrétaire-trésorier

Dr. Fred G. Peet
Pacific Forest Research Centre
Victoria, B.C.
V8Z 1M5

**CANADIAN IMAGE PROCESSING AND PATTERN RECOGNITION SOCIETY /
L'ASSOCIATION CANADIENNE DE TRAITEMENT D'IMAGES ET RECONNAISSANCE DES FORMES**

President/Président

Ching Y. Suen
Department of Computer Science
Concordia University
1455 de Maisonneuve Blvd. W.
Montreal, Quebec
H3G 1M8

Secretary-Treasurer/Secrétaire-trésorier

Morris Goldberg
Department of Electrical Engineering
University of Ottawa
Ottawa, Ontario
K2N 6N5

TABLE OF CONTENTS/TABLE DES MATIÈRES

APPLICATIONS / APPLICATIONS	Page
<i>Computer Graphics and the Fashion Industry</i> J. Nisselson, New York Institute of Technology, Old Westbury, New York	1
<i>Some Implications of Dynamic Structural Analysis</i> J.A. Hoskins and W.D. Hoskins, University of Manitoba, Winnipeg, Manitoba	7
<i>An Encoding Scheme for Presentation Graphics with Animation</i> H.J. Ferch, University of Manitoba, Winnipeg, Manitoba	11
<i>Kinematic and Geometric Modelling and Animation of Robots</i> H.A. ElMaraghy, McMaster University, Hamilton, Ontario	15
COMPUTER GRAPHICS AND ARTIFICIAL INTELLIGENCE / INFOGRAPHIE ET INTELLIGENCE ARTIFICIELLE	
<i>Semantic Network Reasoning for Picture Composition</i> R. Ostrovsky, B.R. Gardner, and M. Holynski, Boston University, Boston, Massachusetts	20
<i>Experiences with Using Prolog for Geometry</i> W.R. Franklin, Rensselaer Polytechnic Institute, Troy, New York; M. Nichols, North American Philips Lighting Co., Bloomfield, New Jersey; and S. Samaddar, P. Wu, Rensselaer Polytechnic Institute, Troy, New York	26
<i>The Inference Machine Laboratory: Graphic Tools for Knowledge Management</i> J.W. Lewis, Martin Marietta Laboratories, Baltimore, Maryland	32
DISPLAY ALGORITHMS / ALGORITHMES GRAPHIQUES	
<i>PORTRAY - An Image Synthesis System</i> D.R. Peachey, University of Saskatchewan, Saskatoon, Saskatchewan	37
<i>An Adaptive Subdivision by Sliding Boundary Surfaces for Fast Ray Tracing</i> K. Nemoto and T. Omachi, NEC Corporation, Kawasaki, Japan	43
<i>Profiling Graphic Display Systems</i> P. Schoeler, Alias Research Inc. and University of Toronto; and A. Fournier, University of Toronto, Toronto, Ontario	49
<i>Using Caching and Breadth-First Search to Speed Up Ray-Tracing</i> P. Hanrahan, Pixar Corp., San Rafael, California	56
USER INTERFACES I / INTERFACE AVEC L'USAGER I	
<i>What are Visual Programming, Programming by Example, and Program Visualization?</i> B.A. Myers, University of Toronto, Toronto, Ontario	62
<i>An Editing Model for Generating Graphical User Interfaces</i> D.R. Olsen, Jr., Brigham Young University, Provo, Utah	66
<i>Automatic Generation of Graphical User Interfaces</i> G. Singh and M. Green, University of Alberta, Edmonton, Alberta	71
ALGORITHMS / ALGORITHMES	
<i>A Fast Algorithm for General Raster Rotation</i> A.W. Paeth, University of Waterloo, Waterloo, Ontario	77

Table of Contents/Table des matières (cont'd/suite)

	Page
<i>A Cel-Based Model for Paint Systems</i>	
T.M. Higgins and K.S. Booth, University of Waterloo, Waterloo, Ontario	82
<i>Design and Experience with a Generalized Raster Toolkit</i>	
A.W. Paeth and K.S. Booth, University of Waterloo, Waterloo, Ontario	91
<i>Graphics Tools in Adagio, A Robotics Multitasking Multiprocessor Workstation</i>	
S.A. MacKay and P.P. Tanner, National Research Council of Canada, Ottawa, Ontario	98
 MODELLING I / MODELISATION I	
<i>Exploiting Classes in Modeling and Display Software</i>	
T. Whitted and E. Grant, University of North Carolina, Chapel Hill, North Carolina	104
<i>Applications of World Projections</i>	
N. Greene, New York Institute of Technology, Old Westbury, New York	108
 HUMAN ANIMATION / ANIMATION DU CORPS HUMAIN	
<i>Animating Human Figures: Perspectives and Directions</i>	
N.I. Badler, University of Pennsylvania, Philadelphia, Pennsylvania	115
<i>The Interactive Specification of Human Animation</i>	
G. Ridsdale, S. Hewitt, and T.W. Calvert, Simon Fraser University, Burnaby, British Columbia	121
<i>Goal Directed Animation using English Motion Commands</i>	
K. Drewery and J. Tsotsos, University of Toronto, Toronto, Ontario	131
<i>Speech and Expression: A Computer Solution to Face Animation</i>	
A. Pearce, B. Wyvill, G. Wyvill, and D. Hill, University of Calgary, Calgary, Alberta	136
<i>Virya - A Motion Control Editor for Kinematic and Dynamic Animation</i>	
J. Wilhelms, University of California, Santa Cruz, California	141
<i>Near-Real-Time Control of Human Figure Models</i>	
W.W. Armstrong, M. Green, and R. Lake, University of Alberta, Edmonton, Alberta	147
<i>Modeling and Animating Three-Dimensional Articulate Figures</i>	
D.G. Cachola and G.F. Schrack, University of British Columbia, Vancouver, British Columbia	152
 MODELLING II / MODELISATION II	
<i>Constraint-Based Modeling of Three-Dimensional Shapes</i>	
P. Prusinkiewicz and D. Streibel, University of Regina, Regina, Saskatchewan	158
<i>The Stochastic Modelling of Trees</i>	
A. Fournier and D.A. Grindal, University of Toronto, Toronto, Ontario	164
<i>Methods for Stochastic Spectral Synthesis</i>	
J.P. Lewis, New York Institute of Technology, Old Westbury, New York	173
<i>Interactive 3-D Modeling with Personal Computers</i>	
R.W. Thornton and G.J. Glass, New York Institute of Technology, Old Westbury, New York	180

Table of Contents/Table des matières (cont'd/suite)

Page

USER INTERFACE II / INTERFACE AVEC L'USAGER II

<i>Psychology and the User Interface: Science is soft at the frontier</i> J.M. Carroll, IBM T.J. Watson Research Center, Yorktown Heights, New York	186
<i>Learning Graphics Programming by Direct Communication</i> M. Tuori and T. Pointing, Defence and Civil Institute of Environmental Medicine, Downsview, Ontario	188

HARDWARE / MATÉRIEL

<i>VLSI and Graphics at the Pixel Level</i> H. Fuchs, University of North Carolina, Chapel Hill, North Carolina	193
<i>Hardware Assistance for Z-Buffer Visible Surface Algorithms</i> K.S. Booth, D.R. Forsey, and A.W. Paeth, University of Waterloo, Waterloo, Ontario	194

ANIMATION / ANIMATION

<i>Eliminating the Dichotomy Between Scripting and Interaction</i> J.F. Schlag, New York Institute of Technology, Old Westbury, New York	202
<i>Survey of Texture Mapping</i> P.S. Heckbert, New York Institute of Technology, Old Westbury, New York	207
<i>Keyframe-Based Subactors</i> L. Forest, D. Rambaud, N. Magnenat-Thalmann, and D. Thalmann, Université de Montréal, Montréal, Québec	213
<i>The Representation of Water</i> G. Wyvill, University of Otago, Dunedin, New Zealand; and A. Pearce and B. Wyvill, University of Calgary, Calgary, Alberta	217

VISION/GRAPHICS INTERFACE / INTERFACE ENTRE LA VISION PAR ORDINATEUR ET L'INFOGRAPHIE

<i>Part Structure for 3-D Sketching</i> A.P. Pentland, SRI International, Menlo Park, California	223
<i>Interfacing Image Processing and Computer Graphics Systems Using an Artificial Visual System</i> J.M. Coggins, K.E. Fogarty, and F.S. Fay, University of Massachusetts Medical School; and Worcester Polytechnic Institute, Worcester, Massachusetts	229
<i>Connected Component Labeling Using Modified Linear Quadrees</i> X. Wang and W.A. Davis, University of Alberta, Edmonton, Alberta	235
<i>Asterisk*: An Extensible Testbed for Spline Development</i> J.R. Gross, T.D. DeRose, and B.A. Barsky, University of California, Berkeley, California	241
<i>Graphical Applications of L-Systems</i> P. Prusinkiewicz, University of Regina, Regina, Saskatchewan	247
<i>Fractals, Computers and DNA</i> P. Oppenheimer, New York Institute of Technology, Old Westbury, New York	254
<i>A Knowledge-Based Approach to Computer Vision Systems</i> M.D. Levine and W. Hong, McGill University, Montréal, Québec	260

Table of Contents/Table des matières (cont'd/suite)

	Page
REMOTE SENSING AND GEO INFORMATION SYSTEMS / TELEDETECTION ET SYSTÈMES D'INFORMATION GÉOGRAPHIQUES	
<i>Integration of Remotely Sensed Data and Geographic Information Systems</i> D.G. Goodenough, Canada Centre for Remote Sensing, Ottawa, Ontario	266
<i>Image Segmentation Based on Color and Texture Gradient</i> P.T. Nguyen, IBM France, Paris, France	267
<i>Map/Image Congruency Evaluation Knowledge Based System</i> G.W. Plunkett and D.G. Goodenough, Canada Centre for Remote Sensing, Ottawa, Ontario; and M. Goldberg, University of Ottawa, Ottawa, Ontario	273
<i>A Context Based Technique for Smoothing of Digital Thematic Maps</i> B. Yee and D. Turpin, MacDonald, Dettwiler and Associates Ltd., Richmond, British Columbia; and E. Kenk and M. Sondheim, B.C. Ministry of Environment, Victoria, British Columbia	279
<i>Principe de codage visuel de la couleur appliqué à des images satellitaires</i> M.-J. Lefevre-Fonollosa and H. Cruchant, Centre national d'études spatiales, Toulouse, France	284
<i>A File Organization Scheme for Polygon Data</i> C.H. Hwang and W.A. Davis, University of Alberta, Edmonton, Alberta	287
ROBOTICS / ROBOTIQUE	
<i>Mathematical Morphology Applied to Range Image Processing</i> C.C. Archibald, Machine Vision International, Ottawa, Ontario; and S.R. Sternberg, Machine Vision International, Ann Arbor, Michigan	293
<i>Incremental Construction of 3-D Models From a Sequence of Framed Views: Matching Partial Objects</i> S. Xie and T.W. Calvert, Simon Fraser University, Burnaby, British Columbia	300
<i>Image Techniques for the Identification of Depressions and Other Obstacles in Automated Guidance of Roving Robots</i> M. Adjouadi, University of Hawaii at Manoa, Honolulu, Hawaii	307
<i>A Computational Theory of 3D Shape Reconstruction From Image Contours</i> P. Liang and J.S. Todhunter, University of Pittsburgh, Pittsburgh, Pennsylvania	313
PERCEPTION AND COMPUTATIONAL VISION / PERCEPTION ET LA MATHÉMATIQUE DE LA VISION PAR ORDINATEUR	
<i>Selection and Use of Image Features for Segmentation of Boundary Images</i> D. Walters, University at Buffalo (SUNY), Buffalo, New York	318
<i>Cortical Representation of Texture Primitives</i> A.E.J. Walford and M.E. Jernigan, University of Waterloo, Waterloo, Ontario	325
<i>Speeded Phase Discrimination: Evidence for Global to Local Processing</i> J.M. Barr, University of Oxford, Oxford, United Kingdom	331
<i>Correspondence in Apparent Motion: Defining the Heuristics</i> M. Green, York University, North York, Ontario	337
<i>Three Processing Characteristics of Texture Discrimination</i> T.M. Caelli, University of Alberta, Edmonton, Alberta	343

Table of Contents/Table des matières (cont'd/suite)

Page

IMAGE PROCESSING AND PATTERN RECOGNITION / TRAITEMENT D'IMAGES ET RECONNAISSANCE DES FORMES

Parallel Architectures for Machine Vision

S.L. Tanimoto, University of Washington, Seattle, Washington 349

Determining Displacement Fields Along Contours From Image Sequences

P. Bouthemy, IRISA/INRIA, Rennes, France 350

Edge-Only Matching Techniques in Robot Vision

S. Nagendran and T.M. Caelli, University of Alberta, Edmonton, Alberta 356

A Method of Learning Rules from Uncertain Data Applied to the Computer Vision Problem

D. Hutber and P. Sims, British Aerospace PLC, Bristol, England 361

Sequential Estimation of Boundaries in Texture Images

D. Brzakovic, University of Tennessee, Knoxville, Tennessee;
and A. Liakopoulos, System Dynamics Inc., Gainesville, Florida 366

Detection of Specularities in Colour Images Using Local Operators

P.N.E. Pellicano, Simon Fraser University, Burnaby, British Columbia 370

APPLICATIONS: OTHER / AUTRES APPLICATIONS

Coupling Visual and Dynamic Features to Study Handwritten Signatures

J.-J. Brault and R. Plamondon, Ecole Polytechnique de Montréal, Montréal, Québec 375

Detecting Glass Fibers Using Computer Vision

A.S. Malowany, M.D. Levine, M.R. Kamal, R.Kurz, and A.-R. Mansouri, McGill University, Montréal, Québec 380

Reconstruction and Display of The Retina

K.R. Sloan, Jr., D. Meyers, and C.A. Curcio, University of Washington, Seattle, Washington 385

Optical Character Recognition of Touching Characters

S. Shlien and K. Kubota, Communications Research Centre, Ottawa, Ontario 390

Contour Line Region Segmentation

L. O'Gorman and G.I. Weil, AT&T Bell Laboratories, Murray Hill, New Jersey 396

COMPUTER GRAPHICS AND THE FASHION INDUSTRY

Extended Summary

Jane Nisselson

Computer Graphics Laboratory
New York Institute of Technology
Old Westbury, New York 11568

Abstract

This paper examines the role of computer graphics and new media technologies in the fashion industry. The three phases of computer graphics application are:

- (1) *Design/Visualization*,
- (2) *Pattern Creation/Manufacturing*, and
- (3) *Presentation/Promotion*.

This paper focuses on the third application, illustrating some of the new directions in computer graphics and fashion with examples of the author's work in fashion videos and interactive systems.

KEYWORDS: fashion design, pattern CAD/CAM, stereoscopic, fashion video.

Introduction

Computer graphics has made its impact on most fields which entail design and imaging: movie and television special effects, computer-aided design and manufacturing, computer-aided engineering, molecular design, medical imaging, seismic analysis for oil prospecting, as well as on video-games and flight simulators. One design field for which the assimilation and creative use of computer graphics still poses a challenge is the fashion industry. Computer imaging systems are being applied in three areas: *Design*, *Manufacturing*, and *Presentation*.

The fashion industry is built around a process of producing a new look. The fashion business is an unending visual modification of the definition of society and its image. Ironically, its link to one of the newest image making media is not implicit. The motivation for using computer graphics systems is clearly defined only in the manufacturing phase: to obtain the advantages of an automated system. In the design and presentation phases, it is not yet evident that the computer medium can enhance production and creativity. In these areas, traditional methods are neither easily simulated nor improved. This paper gives an overview of some applications of new media technology in the fashion industry. An emphasis is given to the final phase; examples from the author's work are included to demonstrate how computer graphics animations can generate an

immediacy and capture the imagination in a way which is true to fashion.

Design / Visualization

The computer is a natural medium for the mass production of perspective images; this makes it an effective tool for engineering and architectural design. Garments however are characterized by neither rigid surfaces nor simple geometrical construction.

To accurately model a garment requires a data format that can represent flexible materials and the effects of physical forces such as gravity and surface tension. Hierarchical solid modeling operations are one technique for simulating twisting, bending, tapering and other such transformations of objects. "Deformations" [1], a form developed by Alan Barr, can be used to simulate flexible geometric objects made of fabric. This technique obtains a normal vector of an arbitrarily deformed smooth surface that can be calculated directly from the surface normal vector of the undeformed surface and a transformation matrix. The deformations are combined in a hierarchical structure.

Furthermore, from the level of detail of fiber and fur to a pattern on a fabric, modeling a garment requires sophisticated techniques such as texture synthesis and stochastic modeling [2]. Patterns on flat fabric can be texture mapped onto the curved surface of the constructed garment [3]. Also, methods are being created to warp the surface of an analytically defined object [4].

The jacket design for a computer generated character, *User Abuser*, demonstrates a method developed at the Computer Graphics Lab, NYIT, for creating a realistic three-dimensional model of a garment. The jacket was modeled as a non-closed surface defined by polygons in a mesh format. This format consists of a list of 3d vertex coordinates, with normals and texture coordinates, topologically connected into a grid with a certain number of rows and columns [5].

One of its most important features is flexible joints at the shoulders and elbows, modeled with flex software by Richard Lundin [6]. These are necessary to correctly deform the surface when rotations are

performed at a joint in a tree structure of a three-dimensional model. Each flexible joint is composed of a single polygonal surface in mesh format which changes shape as the joint is flexed; the joint begins at a particular body part in the model tree and ends at another body part. A bezier spline curve, which defines the centerline of the flexible joint, is created from control points which include the end points of the flexible joint and the coordinates of any joint nodes between the end points. The flex software determines the joint axis, calculates the two end coordinates of the joint, and finally distributes and orients the mesh rows, with respect to the centerline of the flexible joint. This allows the arms to be freely moved, while leaving the garment's seams intact!

The jacket can be rendered with any texture and color and displayed from any point of view [Figure 1]. The appearance of the jacket also depends on lighting which, in this case, creates the jacket's specular surface.

From the vantage of the computer graphics industry, the simulation of three-dimensional garments is a technical problem which will probably gain more attention. In part, this is because the efforts to model and animate the human figure are achieving more and more success. Obviously, this will converge upon the next challenge: designing and animating clothes for this figure.

Without a practical means for representing three-dimensional garments, computer graphics design systems in the industry have been geared towards manipulating two-dimensional designs. The thrust of these systems is to provide (1) an efficient means for visualization and (2) an optimal way to create a design to be readily used in a CAM system.

Textile design is an area related to the fashion industry where computer graphics are widely employed as a design tool. A separate activity from garment construction, textile design requires only two-dimensional representations. Weaving and textile design have long been familiar to computer graphics.

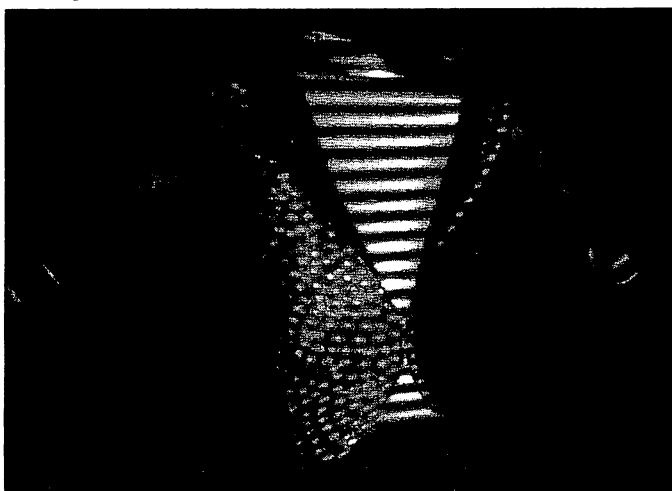


Figure 1 Suit jacket with texture map.

In the early 19th century, Joseph Jacquard developed punched cards as a means to control weaving machines in textile mills. Charles Babbage used these as a model for the cards he used to control the sequencing operations of his calculating machine [7]. Now, there are computer-interfaced looms which can control the pattern and handle complex designs as easily as simple ones. Computer imaging systems can analyze and visualize weave structures. Designs can quickly be scaled, repeated, recolored, and modified without having to laboriously redraw them by hand. The new design can then be produced using computer-interfaced looms.

For example, a textile designer can create textiles with a microcomputer by means of the AVL Loom. Textile designs can be generated in full color with the AVL software automatically calculating an accurate representation of the warp and weft. Via an RS232 connection, the AVL then can control the pattern as the loom is actually weaving it off, draw the pattern with a color plotter, or produce a black and white printout.

The direction of the industry has been to tighten the link between design and manufacturing phases. In part, this is because the most efficient way to give pattern data to a CAM system is to create the garment on a compatible design system. Thus, systems dedicated to the design phase have tended to evolve closely with those for manufacturing.

A system for designing textiles and previewing them on garments has been developed by Computer Design, Inc (CDI). Compatible with traditional design methods, this system is a tool for rapidly visualizing design variations without extensive sample and sketch making. The data describing designs can subsequently be manipulated with a variety of CAM systems. The menu driven system uses an Iris Workstation, from Silicon Graphics, Inc., which has a 68000-based processor, Geometry Engine, and raster subsystem; this system has 1024 x 1024 resolution and 24 bits of color. A more powerful version uses the Iris Turbo which has a 68020-based processor. Fabric designs can be created on the system with its paint program with an electronic tablet or mouse. The system can calculate the number of threads, warp and weft for a given weave (such as twill or oxford) and yarn size. Characteristics such as color, scale, and repeats can be readily manipulated. Fabrics can then be mapped onto garment patterns.

One of the system's features is that designs from other sources, such as samples and sketches, can be scanned in with a camera and then manipulated by the paint system with elaborate functions. Using the paint system and mapping functions, a design from any source can be viewed instantly in a repertoire of colors, patterns or textures.

Although the system works only with two-dimensional images, its functions preserve the dimensional effect created by textures or shadows in a drawn or photographic image of a garment. When

the garment is specified in a new color, a transparency function preserves these visual cues which suggest folds and curves. The texture mapping also takes into account light intensities so that the fabric pattern is realistically mapped onto the garment.

In many respects, the role of computer graphics in the fashion industry's design and presentation phases is similar to its role in movies, television, and advertising. Its use in these areas is not always cost-effective. For lines of clothes which are not mass-produced or mass-marketed, using computer graphics for design alone may be too expensive. Furthermore, cost alone may not be an adequate motive for a designer to make use of the technique. Thus, the role of computer graphics in creating designs in many ways remains subject to the public's demands for aesthetic effects.

In many ways, haute couture has an image which is antithetical to automation and thus does not seem likely to embrace technology in any highly visible phase of its production. It is in ready-to-wear fashion, an industry directed to the "modern" woman, that computer graphics may prove to be in demand. There are some instances of designers who actually reflect the *image* of computer technology in their work rather than using it as a hidden process behind it. The designer, Jurgen Lehl has used computer chips and highly pixellated images as patterns in his individualistic and symbolic textile designs [8]. The fashion designer, Elisabeth de Senneville bases her line on the theme of new images. Dresses, sweat shirts, sweaters often have prints with computer and video images. She explains that when she designs a garment she always asks herself if it could be worn in the year 2001.

Pattern Creation / Manufacturing

In the fashion industry, the adaptation of computer imaging techniques in the area of manufacturing is more prevalent than in its other phases. The overall goal is to automate and manage quality control in all phases of design and production. Computer graphics appear in the form of CAD/CAM systems for digitizing, grading, marking, and sizing patterns in two dimensions. These are formulaic and mechanical operations which do not rely on creative decisions.

Systems are designed to depend on expertise in pattern marking or marker grading -- not computer knowledge. The Lectra System CAD/CAM systems give an example of a highly modular system in which each unit is dedicated to a particular subset of the traditional tasks. Each component has 68000-based processor and disc storage that can range from 256 to 1024 kbytes. Pattern pieces are input on a digitizing table. Next, the pieces can be manipulated -- graded, sized, and markers made -- in real time on a color vector graphic display workstation with a pen and tablet. The pieces can also be "dragged" around and magnified on the display in real time. The plotter unit is used for either single-ply laser cutting or draw-

ing markers on spread fabric. A Winchester-based module with a networking controller provides extra storage. Modules can communicate between different locations over telephone lines by means of a modem.

The next step in manufacturing is a system which can model three-dimensional garments and then unfold them into two-dimensional patterns for subsequent manipulation on a CAM systems. GGT is developing a three-dimensional modeling system that can display rigid models of garments in high resolution (1280 x 1084), using z-buffer algorithms for hidden surfaces, and Gouraud shading. While it may be expensive -- in computer time alone -- to accurately model many of a garment's physical characteristics (such as draping), their impact on the geometric layout must be evaluated and accounted for in the pattern. GGT is currently researching how to reflect engineering concerns including the effects of gravity and warping of a flexible surface when translating the model from three dimensions to a two-dimensional pattern.

CDI is marketing a system for making shoe designs and patterns which is compatible with its design system described earlier. Shoes are characterized by having rigid forms which closely fit the model. Thus the problems posed by flexible surfaces are minimal. A shoe form which is input on a three-dimensional digitizer can be manipulated as a three-dimensional, smooth shaded image. The design is then unwrapped from which point it can go on to standard pattern engineering and CAM systems.

CDI's garment system, under development, also begins with a digitized mannequin. The image can be displayed from different points of view in four windows on the color monitor. The designer draws style lines which the system accurately maps to the surface; changes are updated in all windows simultaneously [Figure 2]. Mirroring functions can automatically reflect designs.

The problems are more complicated for the design of garments whose fabrics and construction do not adhere to the mannequin's shape. At present,

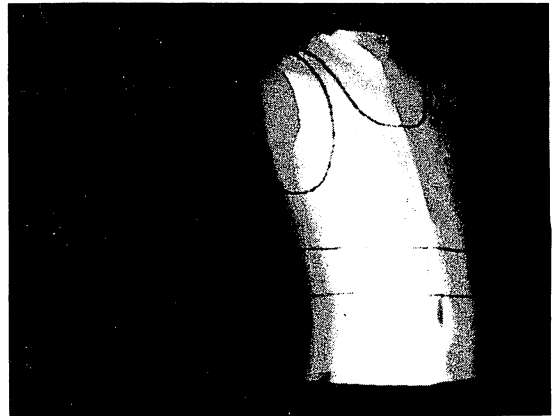


Figure 2 Smoothshaded three-dimensional image of digitized mannequin with pattern lines.

CDI is not attempting complete physical simulation of garments under the assumption that the level of detail may have applications in engineering but is not absolutely necessary in garment industry.

Presentation / Promotion

The technology for new ways of shopping overlaps with the technology for new ways of entertaining, promoting, educating, and informing. In many ways, much as it is in the design phase: it is a tool for visually exploring a database of images, editing and making final selections.

The cosmetics industry is one sector of the fashion industry which explores this. New technologies and cosmetics have a certain compatibility. "Science" is perceived as the means for attaining a variety of technological "miracles" that range from anti-aging products to long-lasting lipsticks. Most people welcome the association of technology with cosmetics because they perceive it as being particularly accurate and scientific. In the past year, "makeup computers" have been introduced by Elizabeth Arden and two Japanese companies, Shiseido and Intelligent Skincare (I.S.).

These systems offer two kinds of services: computerized techniques for (1) makeup application and (2) skin analysis. Basically, the makeup systems use a standard computer graphics technology -- a paint system -- as an innovative way to present the makeup.

All of these companies have systems which provide environments where a trained makeup artist can simulate the application of "electronic makeup" to the customer's face. For instance, Elizabeth Arden's Single Unit System called Sue (refined from its original 3-station system, Elizabeth) houses all of its components in a tall unit which is similar in appearance to a commercial video game. The customer's face is instantly scanned in, by means of a standard video camera, and digitized. The screen's 512 by 512 raster image is divided into four quadrants: one for displaying the initial image of the customer's face and the others, for three different makeup applications. A menu provides palettes for blending suggested colors, selecting brush type and stroke firmness. Software can magnify the face to focus on details. A transparency function maintains the face's original texture while makeup is applied. At the end of the session, a print-out with suggested makeups scrolls out from a slot on the unit's side.

The systems cost each company about \$1 million to develop and thus may not be offered by every cosmetic company. They have had an enthusiastic response. Shiseido, for instance, quintupled its sales, when it introduced its system at Bloomingdales in New York City, in the autumn of 1984 [9].

The appeal of these computer graphics systems is that they allow the customer to interact with the product by means of a simulation. Elizabeth Arden's

marketing department has observed that potential customers are often reluctant to experiment or invest in a new product. With electronic makeup, a customer can experiment not only without being threatened, but without removing the makeup being worn.

The skin analysis system demonstrates the importance of having a human tending the interface between "science and beauty". Based on a personal computer, Arden's skin analyzer scans in the customer's skin with a black and white video camera. The computer constructs a three-dimensional image of the skin surface texture from light intensity information. The data is normalized with additional information such as the customer's age and skin type. Originally, the system assigned a numerical value to the skin type and recommended specific products using a logic system with a hierarchical database of 94 products and regimes. However, an uninterpreted numerical skin rating can alienate a potential customer. The computer thus works as an assistant to the consultant by performing the skin analysis and recommending an overall product line.

The notion of using simulations with which a customer can interact has been explored in the area of fashion as well. The Magic Mirror, designed in Paris by Jean-Claude Bourdier, is an interactive system which uses simulation to create the visual effect for a customer that he is "trying on" clothes. The system wittily updates the traditional interface between customer and garment: the mirror. Set in an environment that is like a darkened dressing room, the Magic Mirror optically combines the reflection of the customer's face with the reflection of a slide projection of an clothing outfit. In the L.S. Ayres store in Indianapolis and Galerie Lafayette in Paris, a trained operator uses a computerized focusing system with a pushbutton interface to project the slide at a scale that fits the customer's figure (size and height). In stores in Japan, the customer simply operates the Magic Mirror himself. Although the system is primarily based on "low" technology, it is easy to imagine the visual database updated so that it is accessed from a videodisc or computer generated.

The video image and fashion are not a novel combination. Often in the form of "news magazines," fashion reportage typically consists of interviews intercut with videotapes of fashion shows which concentrate on good photography of runway presentations. Basically, these presentations are documentaries which often have a canned look. Videotapes are frequently running in various departments of stores. And of course, television broadcasts a barrage of commercials for cosmetics, perfumes and blue jeans. By adding computer graphics and interaction, the customer can gain a fresh look at the product.

The Fizzazz store, designed by Music Video Productions, uses interactive computer graphics systems to present Murjani International's line of CocaCola clothes. The centerpiece of the store is a pair of telev-

ision displays on which a customer can browse through images of the clothes. Each system appears to be simply a television, sheathed in a pared down chrome housing which is mounted on a slim column. Transparent touch sensitive plasma screens cover each of the screens. A computer generated graphical menu is composited with the video images; special hardware was designed by Sony to composite the videodisc (ntsc) and computer graphics menu (rgb). The interface is direct: no keyboards, cursors, mouses or computer languages. The hardware is placed behind the glass windows of CocaCola coolers, off near the dressing rooms.

Customers call up pictures of the clothes which are stored on videodisc by touching menu items on the screen. The images are drawings of the garments depicted in various colors and from different points of view (front and back) and at various level of detail (pocket, cuff, sleeve, etc.). They can be viewed by touching the appropriate selection on the screen. The presentation is geared for a young market which is already used to browsing television for visual stimuli and fashion ideas. Multiple video projection systems display computer graphics animations and stills from the videodiscs on a wall. Visible from the street, the video systems run twenty-four hours a day and the animations are changed weekly. Murjani is gearing the store towards twenty-four hour shopping where the customer electronically looks through the clothes, purchases by means of a credit card, and has the goods delivered the next day. The designers of the system note that it is important that the clothes are available in the store to be handled and tried on.

Like the makeup systems, fashion presentation systems introduce customers to a broader "database" of products than they might otherwise consider -- and give them a tool to manage it. These systems put to use the same shopping techniques exercised without technology: collecting and editing data with the goal of making a purchase.

Developing the notion that interactive computer graphics can provide a medium that can evoke the "presence" of the garment, the author designed a system (at the Architecture Machine Group, Massachusetts Institute of Technology) where a joystick can be used to interactively rotate a garment. This was accomplished by photographing a coat [10] from a series of points of view. The photographs were digitized and laid out in a grid as one picture. This was then displayed in an AED framebuffer which could pan to and zoom up the appropriate image in real time as the viewer moves a joystick to indicate the direction from which he wants to view the garment.

In another project, the same database of multiple images could be manipulated interactively in a stereoscopic computer graphics workstation [11, 12]. The workspace combines a stereoscopic video display with a real space by means of a semi-transparent mirror. Images were placed and moved in the space using an electromagnetic 6 degree of freedom digitizer

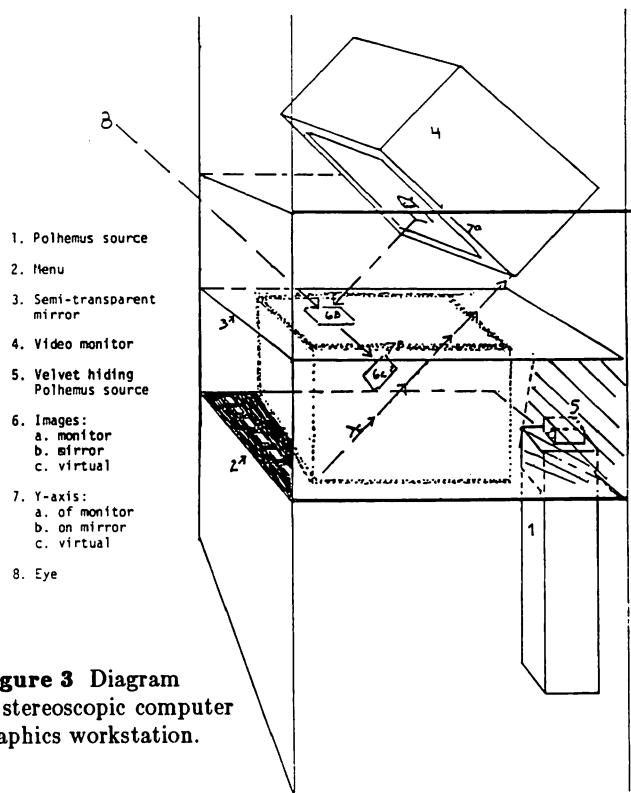


Figure 3 Diagram of stereoscopic computer graphics workstation.

mounted on a small "wand" with a pushbutton mounted in the handle [Figure 3]. The viewer could browse through different views of the garment which he had placed in the workspace [Figure 4].

When the modeling techniques and adequately powerful technology become available, such systems can also be imagined as a tool for the designer to preview a design. Rather than working with a database of digitized pictures, the designer creates a three-dimensional model of a garment which can then be displayed in the workspace from any point of view, at various levels of detail, and with colors and textures defined on the fly.

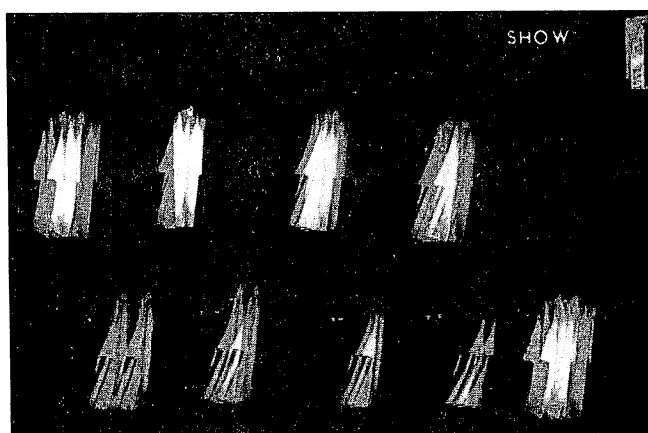


Figure 4 Coat displayed from different viewpoints in workspace. Both left and right images of stereoscopic pair are visible.

SOME IMPLICATIONS OF DYNAMIC STRUCTURAL ANALYSIS

J. A. Hoskins and W. D. Hoskins

Department of Computer Science

University of Manitoba
Winnipeg, Manitoba

1. Introduction.

The interlacement structure exhibited by a woven textile has traditionally, and conveniently, been represented by a binary array. Graphically, this has taken the form of a cartesian grid with cells coloured either black or white [Figure 1]. This binary structural array can, in fact be considered as the product of three matrix factors. These factors are normally also binary matrices and, in most cases, their product arises as a result of conventional matrix multiplication [1]. The factorization process is of great practical interest, since these matrix factors

correspond to parameters for the production of the corresponding structure [2], as well as being of considerable theoretical interest. The development of fast efficient factoring algorithms has been considered and has resulted in the processes described in [3,4].

In analysing an interlacement structure in this way, the data structure is first created using interactive graphical input to colour the cells of the corresponding grid [5,6]. When the data structure is complete, the factorization algorithm is invoked and

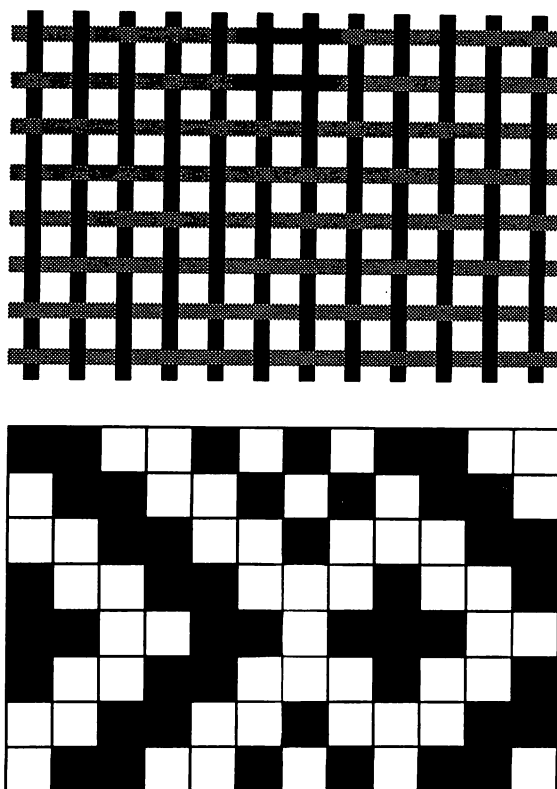


FIGURE 1

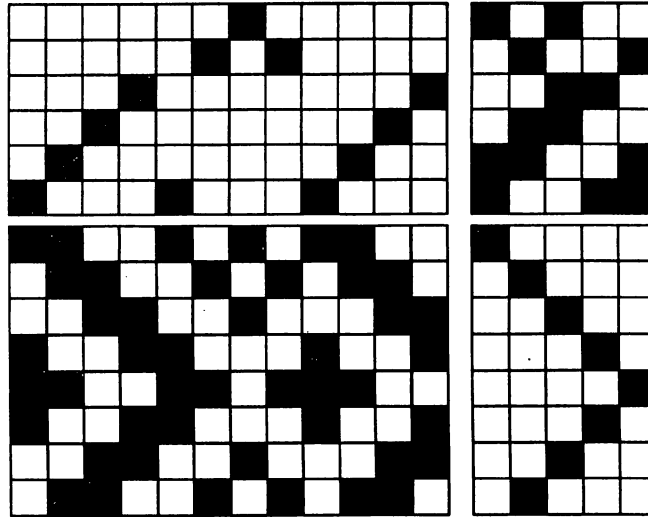


FIGURE 2

the *a posteriori* matrix factors computed. The graphical display is then updated to include their graphical display [Figure 2]. The main advantage to this approach is that data design updates are rapid, since no analysis-processing takes place at the time of design creation. This was of particular importance in previous implementations on small low speed microprocessors, where an emphasis was placed on the design environment [7]. However, a serious disadvantage is that, as the structural array is modified, there is no continuous feedback as to the structural implications of design modifications on the corresponding matrix factors. In this paper, algorithms for performing continuous dynamic factorization in response to incremental data modifications, are considered, along with the ensuing implications for the corresponding graphical display.

2. Dynamic Analysis Model.

The analysis process can be considered to consist of three distinct phases, corresponding to each of the three matrix factors. The first phase requires that all of the columns of the structure array be sorted into distinctness classes [3]. Each distinct column is allocated a separate row in the threading matrix (top left in Figure 2), and all identical columns of the structure array have the single non-zero element in the corresponding columns of the threading array in the same row. The second phase of the algorithm requires that the rows of the structure array be sorted in the same manner, and that each distinct row be allocated a separate column of the shed sequence matrix (bottom right in Figure 2). The third phase involves determining a mapping between the distinct columns and rows of the structure array. This information is recorded in the tie-up matrix (top right in Figure 2), and can normally be generated simply as the intersection of one representative of each distinct column with one representative of each distinct row, without further processing.

The expensive part of this algorithm is the bit-wise comparison of all the columns of the structure array. Although the rows also require a bit-wise comparison, having determined the distinct columns of the array, only one representative of each distinct column set need be considered and the number of elements in each row is greatly reduced. The sorting process can be represented graphically, very effectively, using a *digital trie* [7] for the columns and for the rows.

In constructing the array in Figure 1, initially all of the cells are coloured white, all of the columns are in a single node of the *threading trie*, and all of the rows are in a single node of the *shed sequence trie*. The single node, a leaf in this case, of the *threading trie* contains all of the column indices along with the binary sequence {00000000}, while the single node of the *shed sequence trie* contains all of the row indices along with the binary sequence {0}. As each change is made to the data structure, new branches are added, between the appropriate existing leaf and the new point of disagreement between two previously identical columns. The leaf column indices and binary sequences are also updated. After updating the *threading trie*, the *shed sequence trie* is modified. At each stage, the binary sequences contained in the leaves of the *shed sequence trie* correspond to the columns of the tie-up matrix. The digital tries corresponding to the structure array of Figure 1 are given in Figures 3 and 4.

It can be noted that the traditional binary tree graphical representation consumes a great deal of space, particularly as sequences become more complex. An alternative method of display is given by a *combed trie* representation, where all 'zero branches' are vertical and where 'one branches' appear to the right of any 'zero branches' at the same depth. Figures 5 and 6 show a *combed trie* representation of the structure array of Figure 1.

This data structure provides a means of rapidly updating the matrix factors corresponding to a given structure array in direct response to modifications to that array. It also supports

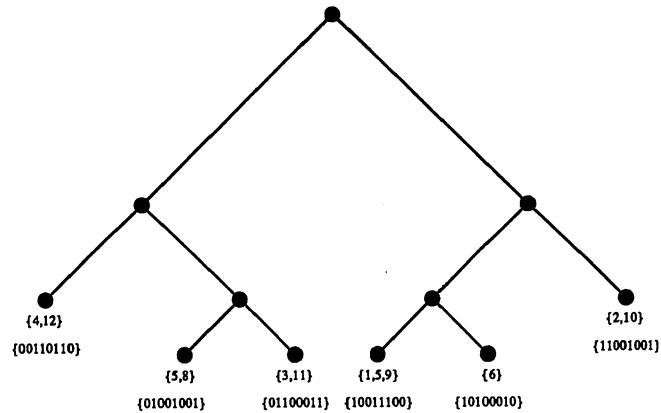


FIGURE 3

an excellent graphical representation of the analysis process which can, in fact, be monitored to determine where changes to the algorithm structure or implementation will result in increased efficiency [8,9].

3. Some Implications for the User Interface.

By using the digital trie data structure, the binary structure array and its corresponding three matrix factors can always be self-consistent. In addition, the rank of the factors can always be minimal and thus, in some sense, optimal. This

pre-supposes, however, that the user will only update the structure array and will never wish to modify any of the three factors. This is definitely not the case. One of the tremendous utilities of such an interactive graphical system is the ability to modify any one of the four components and receive immediate visual feedback as to the implications of this action. This requires that a hierarchy of data elements be maintained by the software, with user-defined cells taking precedence over those set as a consequence of the analysis algorithm. This means that user-defined elements are not moved if, at some stage in the factorization process, their corresponding structure array

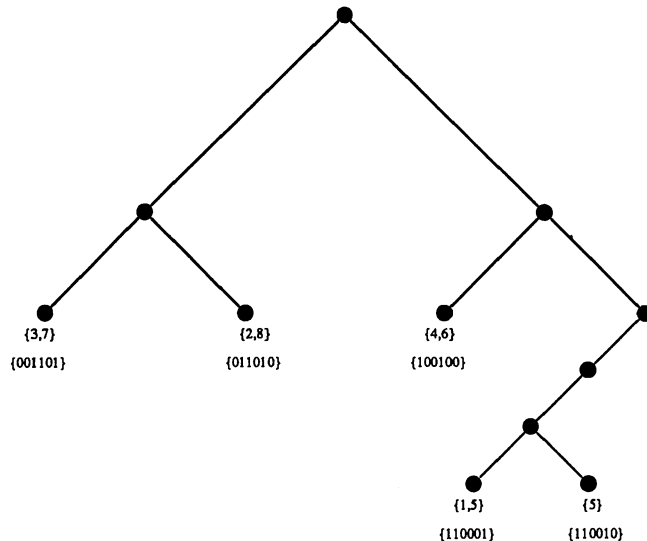


FIGURE 4

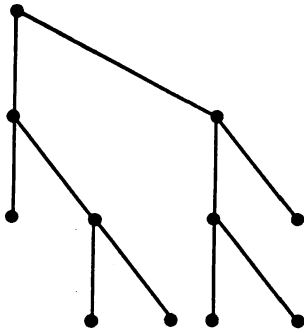


FIGURE 5

column or row becomes identical to another one. While this results in the factors not necessarily being of minimal rank, it does cause the user interface to conform to the design principle of predictability [10].

The consequences of this differential treatment of the user defined components of the partially determined factors, develops in complexity as the design process continues, and of course ultimately begins to limit the choices of the user. Contending with the accruing weight of these limitations places

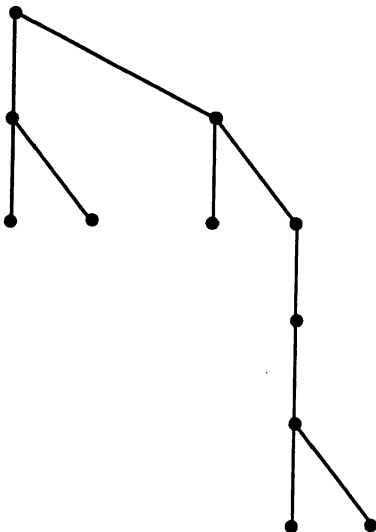


FIGURE 6

an increasing burden on the interactive response capabilities of the system and much further work needs to be done to resolve these problems.

References.

1. J.A. Hoskins and W.D. Hoskins, The Solution of Certain Matrix Equations Arising from the Structural Analysis of Woven Fabrics, *Ars Combinatoria* **11** (1981), 51 - 59.
2. J.A. Hoskins and W.D. Hoskins, Using a Microcomputer for the Design and Analysis of Woven Textiles, 1983 ACM Conference on Personal and Small Computer, San Diego, California, 1983.
3. J.A. Hoskins and W.D. Hoskins, A Faster Algorithm for Factoring Binary Matrices, *Ars Combinatoria* **16B** (1983), 341 - 350.
4. Janet Anne Hoskins, Isonemal Arrays and Textile Computer Graphics, Ph.D. thesis, 1985, University of Manitoba.
5. J.A. Hoskins and M.W. King, Interactive Design of Woven Textiles, Proceedings of the International Computer Color Graphics Conference, Tallahassee, Florida, 1983.
6. J.A. Hoskins and M.W. King, An Interactive Database for Woven Textile Design, Textile Institute Annual Conference "Computers in the World of Textiles", Hong Kong, 1984.
7. Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, Data Structures and Algorithms, (Reading, Mass.: Addison-Wesley, 1983)
8. Marc H. Brown, A System for Algorithm Animation, Proceedings of the SIGGRAPH '84 Conference, Minneapolis, 1984, 177-186.
9. Gretchen P. Brown, Christopher F. Herot and David A. Kramlich, Program Visualization: Graphical Support for Software Development, *Computer* **18** (8), 1985, 27-35.
10. J.D. Foley and A. VanDam, Fundamentals of Interactive Computer Graphics, (Philippines: Addison-Wesley Publishing Company, 1982).

AN ENCODING SCHEME FOR PRESENTATION GRAPHICS WITH ANIMATION

Howard J. Ferch
Dept. of Computer Science, University of Manitoba
Winnipeg, Manitoba, R3T 2N2

ABSTRACT

Bit-mapped raster graphics systems are used in the majority of personal computer systems. As the resolution of these systems increases, and the number of levels of grey-scale, or the number of colours, is increased, encoding of images becomes of greater concern, particularly for interactive systems, or those using animation.

This paper presents a variation of run-length encoding which is faster, but is similar in its degree of space encoding, to run-length encoding. Its application to a menu-driven presentation graphics system is discussed.

One particular advantage of this scheme is its adaptability to the types of animation possible on personal computer raster systems.

KEYWORDS: raster graphics, encoding, animation

INTRODUCTION

Bit-mapped raster graphics systems are very commonly used, particularly on personal computers. These systems typically have a dual-ported frame buffer, which is accessible both to the display hardware, and to the central processor (by mapping the frame buffer into the processor's address space) [FOLEY82]. As memory capacity and speeds go up, and costs go down, it is possible to have higher resolution screen displays, and to display more grey-levels, or colours.

The increased size of the frame buffers means that images are larger, and hence occupy more disk space, take longer to read, and take longer to copy into the frame buffer. This has particular implications to applications in which speed is of major importance, such as those which operate interactively, or

which contain animation.

One system making use of such technology is a menu-driven graphics presentation system being developed for Parks Canada. In this system, an IBM PC-XT equivalent is being used to provide interpretive information to park visitors. Each menu page is displayed in a resolution of 640 by 400 pixels, with 16 colours possible for each pixel. Simple animation sequences are used to present the information. For example, to demonstrate the creation of sinkholes, an animation sequence shows rain falling, the water soaking into the ground and dissolving the gypsum, the ground being undermined, and then the ground collapsing. Since each image takes 128,000 bytes of storage, an encoding scheme is required to reduce the disk space, the main memory space, the disk transfer time, and the display time.

EXISTING ENCODING SCHEMES

A large amount of attention has been paid to the problem of encoding graphical images. (See [HALL79] for an overview of such techniques). However, the majority of such schemes are not adequate for this particular application. Techniques such as Huffman encoding, which use variable length bit strings, are simply too slow for the computing power of a personal computer. Most image creation packages for personal computers (such as Apple's MacDraw [APPLE85]) encode a picture as the objects which were used to create it (e.g. as a sequence of line segments, polygons, etc.). This is the same approach that is used in presentation systems such as Telidon [CSA83]. Again, for real-time animation functions, this approach is too slow. In fact, one of the early objectives of the Parks Canada system was that it was to be much faster than existing museum systems based on Telidon.

A lesser degree of space encoding is provided by run-length encoding, in which an image is stored as a set of tuples,

each containing a colour (or grey-scale) sequence, and an associated repetition factor. Such a technique combines a reasonable level of space compression, with a simple, and hence quick, decoding algorithm. This paper presents a variation of run-length encoding, which is faster, but is similar in the degree of space encoding, and which is particularly adaptable for the type of animation described above.

THE ENCODING SCHEME

Instead of storing a bit pattern (usually a byte or a word), and a repetition factor, one can make use of particular characteristics of the central processor found in the IBM PC. In this processor, (the INTEL 8088)[INTEL81], processor instructions exist to replicate a byte or word through memory, or to copy a string from one location to another. A byte or word pattern to be replicated into the frame buffer may thus be encoded as the machine language instruction sequence required to place that pattern directly into the frame buffer. For example, the following instruction sequence puts the byte containing the hexadecimal value 56 into 250 successive locations of memory (assuming that the appropriate segment registers have been loaded).

```
MOV AL,56H
MOV CX,250
REP STOSB
```

This instruction sequence occupies 7 bytes of memory, and thus we can use 7 bytes to encode 250 bytes of the image. Since this instruction sequence automatically increments an address register to point to the next location in memory (in the frame buffer in this case), we may follow it immediately with another sequence which inserts the next pattern of the image into place, and so on. Thus, the entire image may be encoded as a machine language program, which, when executed, will generate the original image directly into the frame buffer.

For portions of the image which have a large number of very small runs, we can use another sequence in the generated program. Having generated a portion of the image in the frame buffer, it is possible to copy any section of this into a later portion of the image, where the same sequence appears again. On the 8088, an instruction sequence to do this consists of:

```
MOV SI,source offset
MOV CX,length
REP MOVSW
```

This sequence occupies 8 bytes of memory, and also updates the appropriate address

registers. One other instruction sequence is used. This is a variation of the above, which inserts a new string into the frame buffer, by copying it from a copy which is placed in-line in the program, as in:

```
MOV SI,OFFSET LABELx
MOV CX,length
REP MOVSW WORD PTR[DI],WORD PTR CS:[SI]
JMP LABELy
LABELx: DW the string
LABELy:
```

This sequence occupies 11 bytes plus the string length.

In order to apply these sequences, the following algorithm can be used. The original image is stored in main memory, as it will appear in the frame buffer. Starting with the first word of the image, successive words of the image are scanned to see if the image starts with a repeated word. If so, the appropriate machine language code is generated, and the next word of the image is scanned in the same fashion. If the word being examined is not repeated, then the section of the image that has been generated to date is examined to find the largest possible string matching that at the current location. Only if no reasonable length such matching string exists is the third alternative used, and then only long enough to bridge the gap until a repeated word, or a previously encountered string is again encountered.

THE RESULTS

Table 1 summarizes the results, as applied to three sample images from the presentation sequence. The first image is a landscape scene, the second contains a large amount of detail and a large amount of fairly small text, and the third is a map, with a fairly small amount of detail. This table gives the number of occurrences of each of the three instruction sequences, the maximum length of string generated by each, the total number of words generated by each type, the total resulting size of the machine language program, the space reduction factor, (as a percentage of the original size of 128,000 bytes), and the size of the image when encoded using both an object representation, and using run-length encoding. As can be seen from the table, this encoding scheme does use more space than run-length encoding. However, the generation speed is approximately two times faster than using the run-length encoding, due to the lack of overhead spent decoding the tuples of the encoded image. For the three images used, the picture generation averaged 0.3 seconds, while for run-length encoding, the average time was 0.6 seconds.

	IMAGE 1	IMAGE 2	IMAGE 3
	=====	=====	=====
Number of repetition sequences	712	477	648
Max repetition size (words)	1367	530	902
Total number of words created	35008	13396	17824
Number of copied strings	2996	6521	3562
Maximum string size (words)	161	848	159
Total number of words created	27333	49432	44606
Number of new strings	970	868	994
Maximum string size (words)	14	11	11
Total number of words created	1659	1172	1570
Resulting encoded size in bytes	37343	61000	41346
Percentage space occupied	29%	48%	32%
Number of bits per pixel	1.17	1.91	1.29
Size of the run-length encoding	22712	56168	26256
Size of the object encoding	17470	N/A	7870

TABLE 1 - ENCODING RESULTS

REFINEMENTS

One problem exists with the given algorithm. On many of the display adapters using a frame buffer which is dual-ported between the display and the main memory, it is not possible to read from the frame buffer at any desired time. Instead, due to the interactions in the hardware, it is necessary to read the frame buffer only during the retrace intervals, to be sure that the data is correct. Thus the second sequence given above must be modified somewhat. An additional 2 bytes was added to call a subroutine which waits for the retrace to begin. In addition, the maximum length of the move must be limited. Rather than doing this, a refinement to the algorithm was introduced.

The simplest approach was to remove the use of the second sequence (copying already existing sequences). It was felt that this would probably lengthen the encoding somewhat, but the display time would be reduced. In fact, this did not happen. For images which did not contain a large amount of text, the encoding was actually smaller, provided that the encoding was done on a byte, rather than a word level. With this result, the third sequence (generating a new string) was also removed, with the same result. Thus the resulting encoding which is actually used is very simple. Runs of repeating bytes were encoded using only the first sequence, with an additional optimization using a special sequence for runs of length 1 byte, or 2 bytes. For the three images given above, the resulting

encodings had sizes of 37878, 62717, and 38574 respectively. The overall total size across all images was slightly reduced, with no loss of speed.

One other aspect of interest for this encoding approach is its adaptability to other processors. The two major competitors to the 8088 processor are the Motorola 68000 [MOT80], and the National Semiconductor 16000 [NAT83]. While the algorithm can be adapted to both of these processors, it cannot be done in as simple a fashion as it is for the 8088. Neither of these processors provides an instruction which can replicate a value through a range of memory locations. Thus a loop is required. In order to achieve similar compaction levels, this requires the use of an out of line subroutine, with a subsequent loss of speed.

ANIMATION

This encoding scheme is easily adapted to support the type of animation described earlier. To provide animation, a sequence of related images is generated. Then, the above encoding scheme is used to insert the changes from each image to the next in turn into one machine language routine. In many cases, a slower transition from one image to another is desired, such as when scrolling text onto an image, or in having a river change its colour in a given direction, to portray flooding. In these cases, delays are added to the machine language code generated, to achieve the desired rate of speed.

Since dissolves from one scene to the next occur very often, a suite of routines to provide different orderings and timings of dissolves has been created. In the most general case, the user may add a line of any shape and size to an image and request an ordered dissolve from one image to the next moving outwards from the given line, or moving inwards, and he may at the same time specify the speed of movement. Standard top to bottom, left to right, etc orderings are also provided.

Another adaptation allows for repetitive events, such as the blinking of an arrow. In this case, code to insert the arrow, and then to remove it, is generated and then the machine language routine is simply repeatedly executed. Delay sequences are added to achieve the desired rate of blink.

CONCLUSIONS

An encoding scheme for bit-mapped frame buffers which are memory mapped into the central processor's address space has been described. This scheme provides for very fast decoding, while at the same time providing a reasonable level of space encoding.

A complete menu-driven interactive display system for Parks Canada has been constructed using this encoding scheme with the following results. A total of 121 images have been encoded. Of these, 65 primarily contain text (although text and graphics can be freely intermixed), and the remaining 56 primarily display landscape, or map information. In most cases, an image is generated from the previous one using a dissolve sequence. The 65 text images are encoded in 1,166,141 bytes, for an average of 17,940 bytes per image. The 56 graphics displays occupy 1,865,833 bytes, for an average of 33,318 bytes per image. The resulting encodings, including the animation timing delays, and all overhead occupy 19.6% of the original space of the images. The display speed is just slightly reduced from the maximum achievable display speed.

REFERENCES

- APPLE85 Apple Computers, "Inside Macintosh", 1985
- CSA83 Canadian Standards Association, "Videotex/Teletext Presentation Level Protocol Syntax (North American PLPS)", Canadian Standards Association, December 1983.
- FOLEY82 Foley, J.D. and Van Dam, A. "Fundamentals of Interactive Computer Graphics" Addison-Wesley, 1982
- HALL79 Hall, E.L. "Computer Image Processing and Recognition", Academic Press, 1979
- INTEL81 Intel Corporation, "iAPX 86,88 User's Manual", 1981
- MOT80 Motorola, Inc. "MC68000 16-bit Microprocessor User's Manual", 1980
- NAT83 National Semiconductor Corp. "NS16000 Instruction Set Reference Manual", 1983.

KINEMATIC AND GEOMETRIC MODELLING AND ANIMATION OF ROBOTS

Hoda A. ElMaraghy

Department of Mechanical Engineering
McMaster University, Hamilton, Ontario, L8S 4L7

ABSTRACT

At present, most industrial robots are programmed in a teach mode. In the meantime, robots are called upon to perform increasingly complex tasks, which makes programming by teaching rather tedious and cumbersome. There is an increasing need for effective tools to assist in off-line programming of robots and verifying their programmed moves.

A program for modelling and simulating the PUMA 560 and ADEPT I robots has been developed. The kinematic models required for the transformation from task space to robot configuration space, and vice-versa, are briefly outlined. The robot geometric models and their graphical manipulation has been implemented using MOVIE-BYU and PLOT-10 on a Tektronix 4115-B graphics terminal and a VAX 11/730 minicomputer. The developed kinematic and geometric models are used for off-line simulation, animation and verification of robot movements during assembly operations. Sample outputs which illustrate the program capabilities such as shaded images, wire-frame animation, arbitrary point tracing, actual movement envelope and the various menus are included.

KEYWORDS: geometric modelling, robot kinematics, graphical simulation and animation, robot off-line programming.

INTRODUCTION

Robots have recently been gaining popularity and acceptance as an efficient tool for accomplishing various manufacturing tasks. Current applications include material handling, welding, painting, deburring, etc. Robots' greatest potential for increasing productivity is in the field of automated and flexible assembly. The key to their success in this area will be in integrating robots with adequate sensors and providing high level sophisticated programming tools.

One of the major goals of our current research in the Centre for Flexible Manufacturing Research and Development at McMaster University is to develop systems which can automatically synthesize programs for controlling robots performing assembly tasks with sensor feedback. The input to these systems includes:

- a) specification of assembly tasks and goals, and
- b) specification of the initial state of both the robot and world models.

An expert system written in COMMON LISP, currently under development, then uses the knowledge base and production rules to produce a plan for robot motions, and a robot level program in VAL II. The on-line expert system will allow updating of preplanned robot moves in response to sensor input.

Achieving this goal requires several building blocks, including:

- 1) Robot geometric and kinematic models,
- 2) Robot world geometric model, including parts and tasks,
- 3) Sensor models,
- 4) Motion planner,
- 5) Adaptive learning model, and
- 6) Knowledge base and inference engine.

Several research projects are in progress to develop these modules. This paper focusses on the generation of the kinematic and geometric models of the two robots used in the centre, and the links between these models and the rest of the modules mentioned above.

The flexible robot assembly system, installed in the Centre for FMS research and development, consists of two robot work stations, an IRI vision inspection station, a load/unload station, and a computer-controlled Bosch conveyor, with pallets, for material handling. The two robots are a six-axis articulated PUMA 560 and a four-axis ADEPT I scara robot. Both robots are interfaced with force, tactile and vision sensors for real-time adaptive control. The flexible assembly system is used for both research and development projects related to mechanical and electronic assembly.

THE KINEMATIC MODELS

Controlling and programming robots requires analytical models which relate the robot configuration space, expressed in terms of joint variables, and task space normally described in Cartesian coordinates. These models should allow efficient transformation between the two spaces since robots are usually controlled in the joint space while tasks are normally defined in the Cartesian space.

A robot kinematic model consists of two parts:

- 1) Forward Kinematics: which accomplishes the transformation from configuration space to task space, and
- 2) Inverse Kinematics: which transforms representations in task space to configuration space.

Kinematic models may be used to transform positions, velocities and accelerations between the two spaces.

Several methods have been developed in recent years to obtain generalized and efficient solutions to these kinematic models [1-8]. The emphasis has been twofold: a) concise and elegant model representation, and b) fast and efficient solutions to these models. The method and notation used in developing our models is based on those developed by Denavit and Hartenburg [1] as applied to robots by Paul [4]. The results are summarized below.

PUMA 560 ARTICULATED ROBOT

PUMA 560 is a 6-degrees-of-freedom articulated robot where the first three joints are used to place the manipulator wrist in its work envelope and the last three joints are used to orient the wrist. All robot joints are rotational.

The coordinate frames used to derive the kinematic models of the PUMA 560 robot are shown in figure 1. The kinematic parameters are summarized in Table 1.

Table 1: PUMA 560 Configuration Constants

Joint #	Joint Variable	Joint Angles Range (degrees)	
		θ_i min.	θ_i max.
1	θ_1	-160°	$+160^\circ$
2	θ_2	$-\pi - 43^\circ$	$+43^\circ$
3	θ_3	-52°	$+\pi + 52^\circ$
4	θ_4	-110°	$+170^\circ$
5	θ_5	-100°	$+100^\circ$
6	θ_6	-266°	$+266^\circ$

Joint #	a_i (mm)	d_i (mm)	α_i (degrees)
1	0	0	-90
2	431.81	149.09	0
3	-20.31	0	90
4	0	433.07	-90
5	0	0	90
6	0	56.25	0

The reader is referred to [9] for details of the development of both forward and inverse kinematic model solutions of PUMA type robots. This method was closely followed in developing our PUMA 560 kinematic models.

ADEPT I SCARA ROBOT

ADEPT I robot is a 4-degrees-of-freedom manipulator with two rotational joints (#1 and #2) and one rotational and prismatic joint (#3). Figure 2 shows the kinematic model of the ADEPT I and its link and joint parameters. Table 2 lists the arm kinematic parameters.

Table 2: ADEPT I Configuration Constants

Joint # (i)	Joint Variable	Variable Limits		a_i	a_i (mm)	d_i (mm)
		min.	max.			
1	θ_1	-150°	$+150^\circ$	0	425	870
2	θ_2	-147°	$+147^\circ$	0	375	223.5
3	θ_3	-277°	$+277^\circ$			
	d_3	0	203 mm	0	0	0

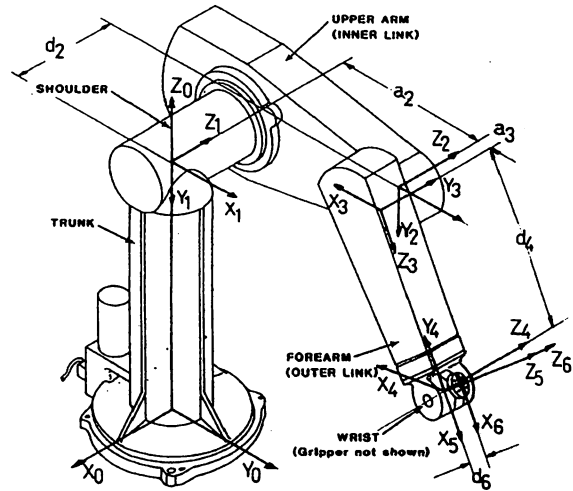


Fig. 1 PUMA 560 coordinate frames and kinematic parameters.

Inverse Kinematics

Using Paul's method, a solution for the inverse kinematics model of ADEPT I was derived.

$$\theta_1 = \tan^{-1} \frac{P_y - P_x \left[(\pm) \sqrt{\frac{1}{A^2} - 1} \right]}{P_y \left[(\pm) \sqrt{\frac{1}{A^2} - 1} \right] + P_x} \quad (1)$$

where

$$A = \frac{P_x^2 + P_y^2 + a_1^2 - a_2^2}{2 a_1 r} \quad (2)$$

$$r = \sqrt{P_x^2 + P_y^2} \quad (3)$$

and the (+)ve and (-)ve signs in equation (1) refer to the right and left hand robot configurations respectively.

$$\theta_2 = \tan^{-1} \left[\frac{P_y \cos \theta_1 - P_x \sin \theta_1}{P_y \sin \theta_1 + P_x \cos \theta_1 - a_1} \right] \quad (4)$$

$$\theta_3 = 2 \tan^{-1} \left[\frac{n_y \cos(\theta_1 + \theta_2) - n_x \sin(\theta_1 + \theta_2)}{n_y \sin(\theta_1 + \theta_2) + n_x \cos(\theta_1 + \theta_2)} \right] \quad (5)$$

where

$$n_x = \cos \theta_3 (\cos \theta_1 \cos \theta_2 - \sin \theta_1 \sin \theta_2) + \sin \theta_3 (-\cos \theta_1 \sin \theta_2 - \sin \theta_1 \cos \theta_2) \quad (6)$$

$$n_y = \cos \theta_3 (\sin \theta_1 \cos \theta_2 + \cos \theta_1 \sin \theta_2) + \sin \theta_3 (-\sin \theta_1 \sin \theta_2 + \cos \theta_1 \cos \theta_2) \quad (7)$$

$$d_3 = d_1 + d_2 - P_z \quad (8)$$

where the actual linear displacement of joint 3 is equal to $d_3 - d_3'$ and d_3' is a physical offset ($d_3' = 215$ mm).

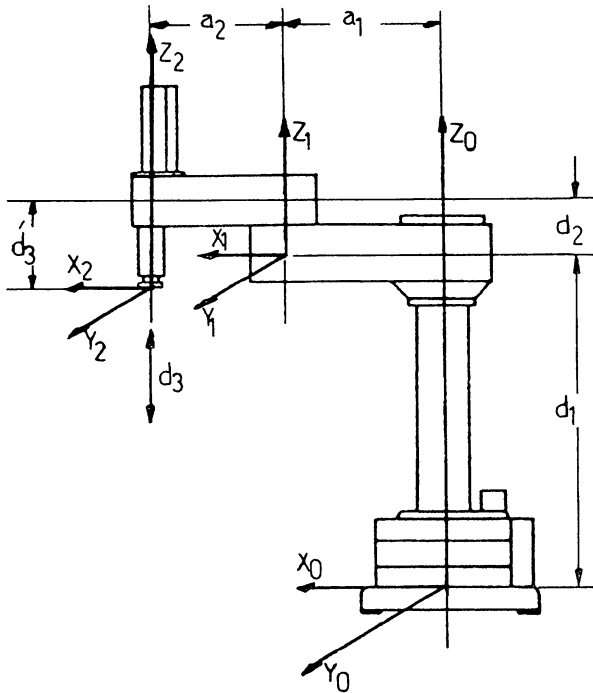


Fig. 2 ADEPT I coordinate frames and kinematic parameters.

Forward Kinematics

The forward kinematics solution defines the Cartesian coordinates of the robot end effector in terms of the robot joint variables as follows:

T_3 = orientation matrix

$$T_3 = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \quad (9)$$

where

$$n_x = \cos(\theta_1 + \theta_2 + \theta_3) \quad (10)$$

$$n_y = \sin(\theta_1 + \theta_2 + \theta_3)$$

$$n_z = 0$$

$$o_x = \sin(\theta_1 + \theta_2 + \theta_3) \quad (11)$$

$$o_y = -\cos(\theta_1 + \theta_2 + \theta_3)$$

$$o_z = 0$$

$$a_x = 0, \quad a_y = 0, \quad a_z = -1. \quad (12)$$

The Cartesian coordinates are given by:

$$P_x = a_1 \cos \theta_1 + a_2 \cos(\theta_1 + \theta_2) \quad (13)$$

$$P_y = a_1 \sin \theta_1 + a_2 \sin(\theta_1 + \theta_2) \quad (14)$$

$$P_z = d_1 + d_2 - d_3 \quad (15)$$

The closed form solutions of the forward and inverse kinematic models of the PUMA 560 and ADEPT I were implemented in FORTRAN 77 and linked with a robot geometric modeller and simulator called ROBOT. The Cartesian and Joint Coordinates, calculated using the developed models, were compared with those measured using the two robots in all eight quadrants. The maximum absolute errors were in the range of 0.0028% and 0.06%.

GEOMETRIC MODELLING

Robot motions are not easy to visualize or verify using the mathematical models only. Interactive graphical simulation of robot motion is a powerful tool for verifying robot programs off-line and evaluating the robot interaction with surrounding objects. It provides for rapid error checking and assists in laying out the robot work cell.

The geometric modelling system MOVIE-BYU was used to generate robots and world models. Wire frames were generated from the geometry data base and used for robot animation to increase display speed. This procedure and the user's menus were developed using GKS PLOT 10 and implemented on a VAX 11/730 minicomputer and a Tektronix 4115B colour display. This approach significantly reduces the overhead normally required by MOVIE-BYU, and maintains the rich details of the graphical model while producing relatively fast images for dynamic display.

PROGRAM OVERVIEW

ROBOT is an interactive program which performs graphical simulation of the kinematics of arbitrary open chain linkages such as robots. ROBOT is primarily menu-driven. All functions are selected from menus displayed on the right hand side of the screen. Keyboard input is only required when parameter values or file names are needed within various functions. Menu selection is done via keyboard, thumbwheels and/or graphics tablet. ROBOT utilizes geometric files created using MOVIE-BYU and kinematic configuration files corresponding to each robot. Currently both PUMA 560 and ADEPT I have been modelled geometrically and kinematically. The program performs the following functions:

- 1) Reads geometric and kinematic data files corresponding to the selected robot.
- 2) Allows the user to create or update the kinematic parameters of the robot.
- 3) Allows the user to specify the geometry and dimensions of end effectors and grippers.
- 4) Accepts the Cartesian coordinates of the robot end effector, solves the inverse kinematic problem, produces the corresponding joint coordinates and displays the resulting robot configuration.
- 5) Accepts joint coordinates, solves the forward kinematics problem, produces the Cartesian coordinates and displays the corresponding robot configuration. Joint parameters may be entered as absolute or relative values.
- 6) Moves the robot to a new configuration either instantly or in a sequence of successive configurations for animation. The number of steps provided by the user is used to create equal step sizes for each joint. Frames may either be left on the screen, resulting in a superimposed sequence of images, or cleared before the next one appears.
- 7) Allows any point(s) in the geometry to be distinguished and used to trace a path(s) in space during robot animation. This feature is useful in

verifying the path traced by a tool or end effector and in visually checking the collision of the selected part of the robot arm with other objects in the workspace. Vertical walls from the selected point trace may also be viewed to further clarify the 3-D path and work envelope.

- 8) Allows the user to alter the view of the world. The 3-D graphical representation produces perspective or parallel projection according to the user's selection. The user's point of view is altered by rotating the robot coordinate system (about x, y or z axes) and centering the image on the display screen.
- 9) Displays in the upper right hand corner of the screen all selected options in the form of ON and OFF switches which can easily be toggled to change the corresponding status.
- 10) Continuously displays and updates the value of robot joint variables and highlights those violated by a given robot move.
- 11) Produces stereo pair for hard copy by generating 2 images of the current view. The first image is a perspective projection from the left eye point of view and the second is from the right eye point of view, which is reflected about the vertical axis in the projection plane. When both images are made into hard copies, they result in a stereo image when viewed simultaneously with the aid of a small plain mirror.

Sample outputs are included to illustrate the various options and capabilities of the ROBOT program.

ROBOT GEOMETRIC MODEL BUILDING

The ROBOT program is a general purpose simulation and animation program capable of simulating any open chain type mechanism. The type of robot, its geometry, kinematics and physical constraints are defined in separate files. Therefore, a library of different robot models may be created for future use. Currently, only the PUMA 560 and ADEPT I robots have been completely modelled in our system.

Each robot is represented as an assembly of robot links. Each link is constructed as a separate object using MOVIE-BYU. The reference coordinate frame for each link is positioned at the link joint according to the Denavit and Hartenberg convention. Therefore, the robot shape and dimensions are modelled as a collection of polygonal surfaces or lines in 3-D space. The geometry consists of Cartesian coordinates in $R \times R \times R$ (called points or nodes) along with a connectivity relation on the nodes which is used to determine which lines are to be drawn in a wire frame model. Any redundant lines produced by the connectivity relation will be deleted and removed by the optimizing module in ROBOT as soon as the robot geometry file has been read. Robot geometry generated in this fashion is stored in files and accessed later by ROBOT. Homogeneous transformation matrices describing the relative translation and rotation between adjacent link coordinate frames are used to transform the modelled links into a complete robot configuration [10].

CONCLUSIONS

The developed kinematic and geometric models of the two robots used in our research program, namely PUMA 560 and ADEPT I, were described. The interactive graphics simulation program, ROBOT, proved very useful in displaying and verifying robot moves. This program will be integrated with a robot task planner which is currently being developed at McMaster

University. It will certainly be very useful in checking out the robot programs produced by the robot tasks planner based on geometric and functional reasoning to achieve the stated goals. This planner is being developed in common LISP and the main application domain is mechanical assembly. Other modules are being added to ROBOT to model objects in the workspace.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the financial support of the Canadian Natural Sciences and Engineering Research Council. Mr. L. Hamid programmed the majority of the ROBOT package. Messrs. D. Brown, S. Payandeh, L. Shauzhong and B. Johns have also contributed to the programming and kinematic modelling.

REFERENCES

1. Denavit, J. and Hartenburg, R.S., 1955, "A Kinematic Notation for Lower- Pair Mechanisms Based on Matrices", ASME Journal of Applied Mechanics, vol. 22, June 1955, pp. 215-221.
2. Duffy, J., 1984, Analysis of Mechanisms and Robot Manipulators, Edward Arnold, London.
3. Featherstone, R., 1983, "Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles", The International Journal of Robotics Research, vol. 2, No. 2, pp. 35-45.
4. Paul, R.P., (1981), Robot Manipulators: Mathematics, Programming and Control, MIT Press.
5. Tsia, L.W. and Morgan, A.P., 1984, "Solving the Kinematics of the Most General Six- and Five-Degree-of-Freedom Manipulators by Continuation Methods", ASME Paper No. 84-DET-20.
6. Yuan, M.S.C. and Freudenstein, F., 1971, "Kinematic Analysis of Spatial Mechanisms by Means of Screw Coordinates. Part 1 - Screw Coordinates", ASME Paper No. 70-MECH-13.
7. Elgazzar, S., 1984, "Efficient Solution for the Kinematic Positions for the PUMA 560 Robot", ERG-972, NRCC No. 23952.
8. Lee, C.S.G., 1982, "Robot Arm Kinematics, Dynamics and Control", IEEE Trans. Comput., vol. 15, no. 12, pp. 62-80.
9. Paul, R.P., Shimano, B. and Mayer, G.E., 1981, "Kinematic Control Equations for Simple Manipulators", Trans. of Systems, Man and Cybernetics, IEEE, pp. 449-455.
10. Foley, J.D. and Van Dam, A., 1983, Fundamentals of Interactive Computer Graphics, Addison-Wesley.

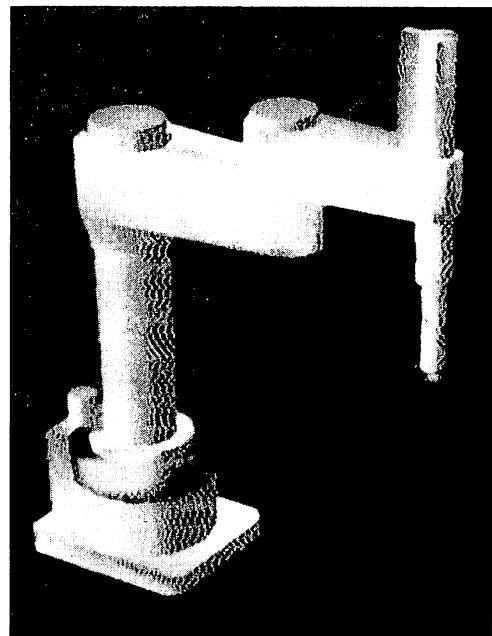
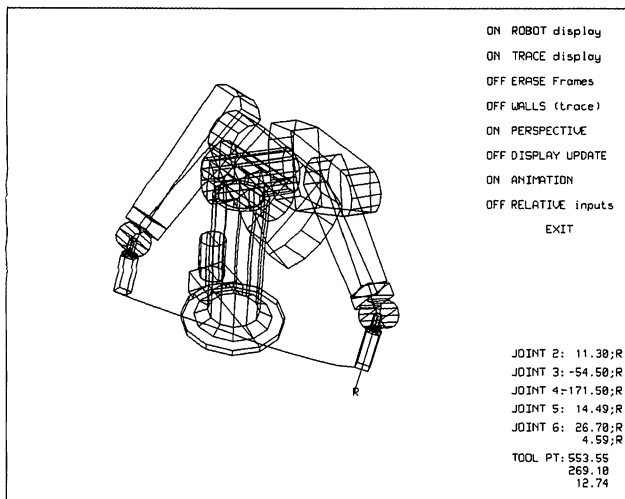
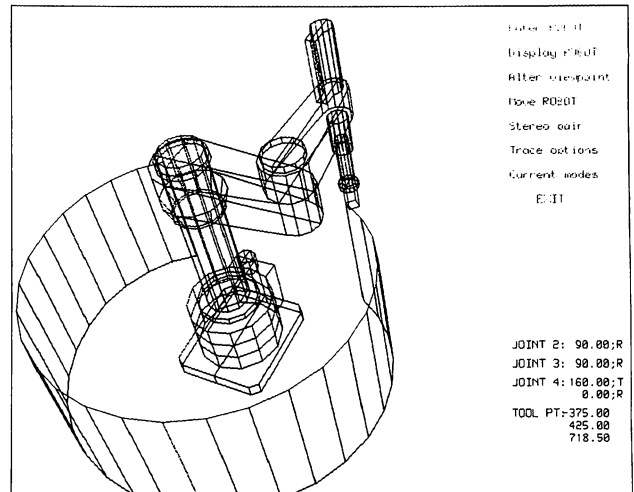
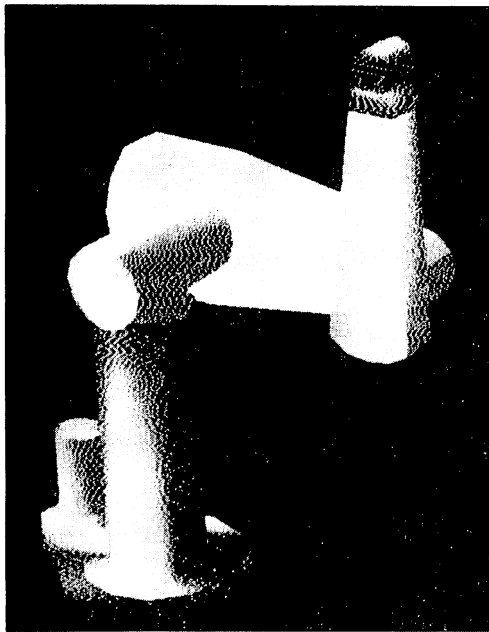


Fig. 3 Examples of graphical outputs produced by ROBOT.
Actual displays are in colour

SEMANTIC NETWORK REASONING FOR PICTURE COMPOSITION

Rafail Ostrovsky
Computer Science
Department

Brian R. Gardner
University Professors
Program

Marek Hołyński
Computer Science
Department *

Boston University, Boston, MA 02215

ABSTRACT

To design an intelligent graphics system, conceptual information has to be represented and reasoned about. This paper explores knowledge representation schema, tools and techniques which are necessary for creating such a system. We will present a system that allows us to relate the meaning of a picture to its graphic representation. Two major blocks of the system are responsible for abstract reasoning about a picture and for picture composition respectively. We will show how a Semantic Network formalism and Semantic Network-based reasoning can be employed to allow abstract reasoning about a picture. How based on conceptual level information, the system reasons about appropriate scene composition and image generation. A smooth transition from conceptual level information to the actual picture composition is also achieved. We introduce a simple environment in which we have tested our approach. Within the constraints of this environment our system reasons about abstract non-visual concepts, decides which physical objects should be displayed, and renders them into an image.

KEYWORDS:

Intelligent graphics system; Knowledge representation; Temporal reasoning; Scene composition.

1. Introduction

Current graphic systems place the tasks of scene composition and object formation on the user. Their performance is limited to the representation of an image that was fully determined by a programmer. The situation would improve when, instead of burdensome testing of different versions of an image, the user could obtain some assistance in deciding about

the user could obtain some assistance in deciding about graphic presentation from an intelligent computer graphics system.

In order for a system to provide such assistance, it should have some knowledge about the nature of the objects which user wants to display. In many cases, the user may wish to give only abstract specifications of the picture he wants to see. How does a system decide **what** to display and in what **form** it should be displayed? What kind of knowledge representation and reasoning mechanisms do we need to cope with this problem? How does one go from abstract specification of the picture to the actual image being generated? In this paper we will try to answer some of these questions and present a design of a system which is responsible for the appropriate picture composition and generation.

To create an intelligent graphics system, tools must first be developed which integrate techniques from the fields of Artificial Intelligence and Computer Graphics in a coherent manner [2]. Our system consists of two modules. Reasoning and determination of the conceptual specifications for the picture is facilitated by the Semantic Network Processing System (SNePS) [4]. SNePS allows us to create the conceptual knowledge base in the form of a semantic network and is capable of including rules of inference in the same representation. It decides on the appropriate meaning of the picture and on the objects which should therefore be included in the picture. SNePS passes this information to a graphics module (Graflisp [1]) along with some set of restrictions for the picture composition.

A graphics module is responsible for object generation, transformation and display rendering. Graflisp maintains knowledge of object structures and of inter-relationships between objects. Objects have a special class inheritance. This allows objects to be comprised of any forms which can be functionally defined, rather than limiting them to be built-up from a polygonal base. Users may define and link their own classes to the classes which are currently defined (polygons, spheres, surfaces of revolution, etc.). In addition to the smooth shading,

* Also, Center for Advanced Visual Studies, Massachusetts Institute of Technology, Cambridge, MA 02139.

realism and perceptual clarity is enhanced by the use of color or texture patterns on the object's surface. The system can reference both the color and texture mapping functions associated to an object at render-time. This allows either the mapping of images (from a camera or a previous image) or of a functionally defined artificial texture onto an object's surface (see Fig. 1).

2. Conceptual level

Since we want to deal with various concepts, we have to **represent** them in our system. Therefore, we use a semantic network in which every concept is represented by a node. Such a semantic network formalism has been implemented by S. C. Shapiro [4]. The arcs between concepts represent relations of one concept to another concept. We can talk about the *distance* from one concept to the other, where, the shortest path via some set of arcs as an intuitive "closeness" of two concepts. In general, the meaning of every concept (i.e. node) in the network is defined in terms of the rest of the network.

We will differentiate between abstract concepts and concepts of some physical objects. Whenever a user refers to some abstract concept, a correspondence should be found between this abstract concept and some physical object. To achieve this correspondence the domain-specific knowledge and "distance" factors are used. If the abstract concept corresponds to several physical objects, the system queries the user and stores the answer as a default choice for future reference. Later, if the system is faced with the same choice, the previous decision is made. The user, of course, may override the default.

Sometimes, in addition to the objects directly specified by the user, we want to display some objects which are **semantically** appropriate in the image. For example, if somebody wants to see a drawing of the president's home, he may expect to see related objects in the scene, such as the president in front of the White House. Thus, at a present time, the system should display Reagan in front of the White House. However, if we ask the system to draw the same picture five years later, it will not be Reagan. Two different kinds of inference are present. A **semantic inference** tells us that when we want to see a picture of the White House, it may be appropriate to display the president in front if it. A **temporal inference** tells us that at the present time the president of U.S. is Reagan. We will examine both semantic and temporal reasoning in our system.

3. Puppets World

To study what type of representation and inference is necessary for integrating conceptual information with the actual picture generation we needed to have a domain which does not require a large body of common-sense knowledge. It should be rich enough to have several abstract concepts, several physical objects and in which temporal reasoning can be tested. We have chosen *Puppets World*†. In Puppets World several puppets live in several houses. The system knows a physical description of every puppet as well as a correspondence of each *concept of a puppet* to its 3-D graphical "body". First, we create concepts (i.e. nodes) of several puppets. We tell to the system that every puppet has a place to live and a place to work. Then, we introduce concepts of a *home* and a *workplace*. We assume that during every day of the week each puppet goes to his corresponding work and is at home during any other time. Thus, if the system needs to know who is present at a certain time and a certain location, the temporal reasoning mechanisms will have to be employed.

We give the following information to the system: Ron, Bill and Jane are puppets. There are two locations: the White House and a Regular House. Bill and Jane live at the Regular House. Jane works at home and Bill works at the White House. Ron lives and works at the White House.

We want our system to be able to generate pictures from the following types of queries: "Show me the house of Bill"; "Show me the home of Ron"; "Show me a picture of a puppet and a house"; "Show me a workplace of Jane", etc. We would like the system to decide what various abstract concepts (like home and a workplace) correspond to. We also want our system to display appropriate puppets with every location if they are at this location at the given time. For example, we want the system to display a regular house and Jane if we ask "Show me the home of Bill" during Bill's work-time.

4. Semantic Inference

After the "Puppets World" data is given to the system, we also have to tell the system about the relations among different concepts. The system knows in which house which puppet lives. We can now proceed to specify what the concepts of "workplace" and "home" mean:

† This name was given as an analog the the well-known Blocks World.

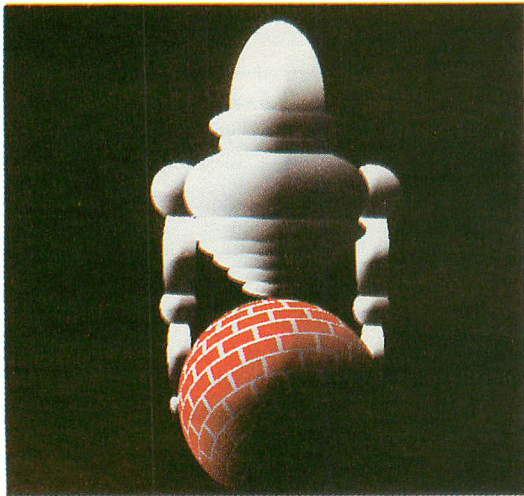


Figure 1. Graflisp example of color mapping, light modelling and hidden surface removal.

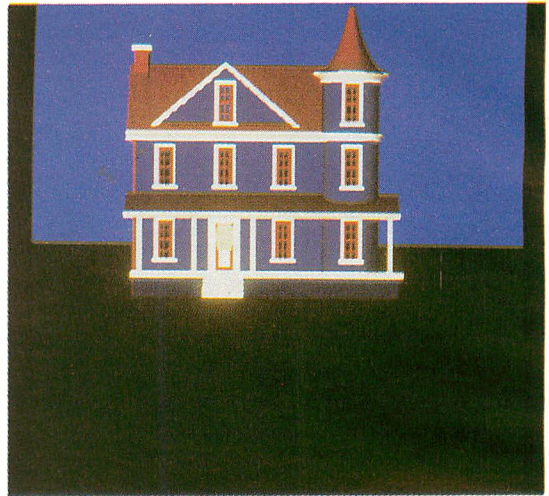


Figure 2. System's response to a request to display a Regular House.

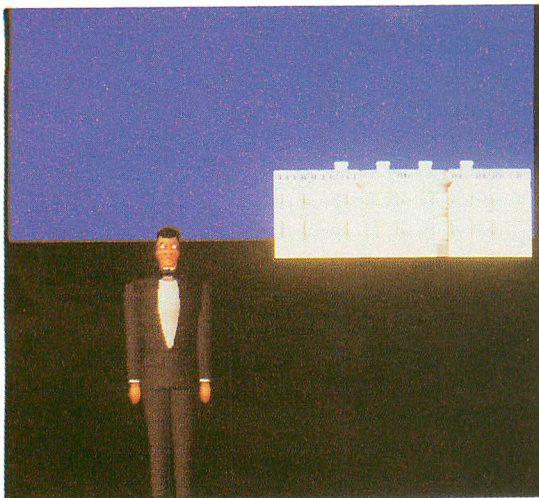


Figure 3. Situation at the White House during off-hours.

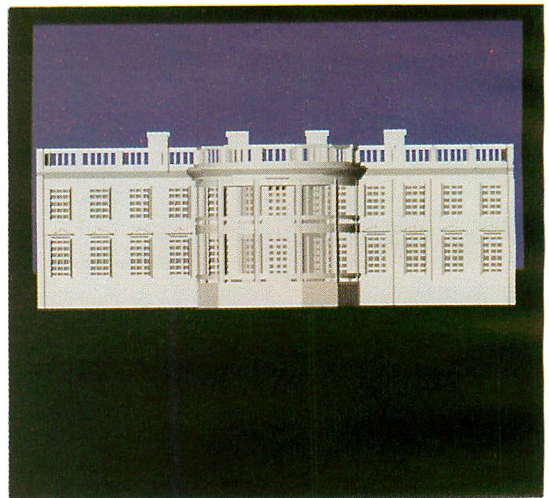


Figure 4. System's response to a user's request to display the White House.

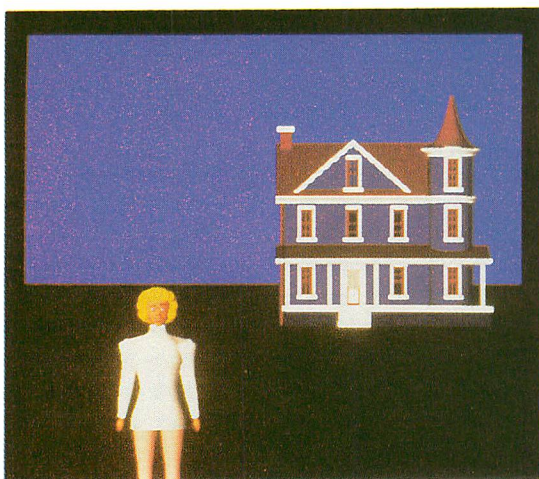


Figure 5. Situation at the home of Bill during business hours.

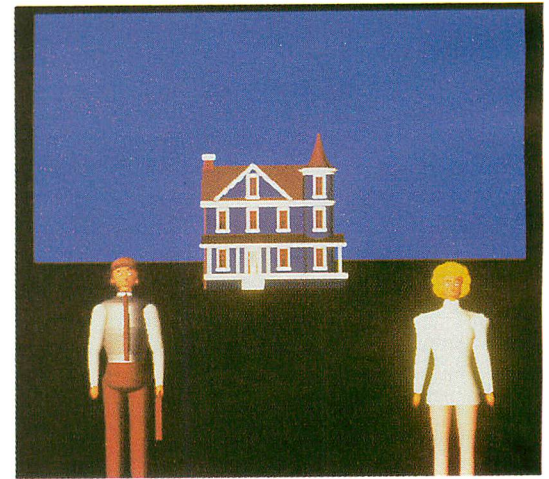


Figure 6. Situation at the workplace of Jane during free time.


```

; For all x
;   if x is a puppet
;   then for all y
;     if y is a house where x lives
;     then y is a home of x.

(build avb $x
  ant (build member *x
        class puppet)
  cq (build avb $y
      ant (build verb live
            actor *x
            place house
            placeName *y)
      cq (build verb live
          actor *x
          place home
          placeName *y)))

```

Thus, the system looks for matches of proposition represented by the rule node **m14** and asserts the proposition under **m15** with appropriate bindings for **x** and **y**. A similar rule is created to represent the fact that a place where puppet works is his workplace:

```

; For all x
;   if x is a puppet
;   then for all y
;     if y is a house where x works
;     then y is a workplace of x.

```

Since the system has the initial knowledge of places where people *live* and places where people *work*, it will be able to deduce what a reference to somebody's *home* or *workplace* mean. For example, imagine that we ask the system to display a workplace of Jane. The system will first check if Jane is a puppet (which is given), then it will look for a place where Jane works and bind value of **y** to the workplace of Jane. The graphics package will then proceed to display a house, which is where Jane works, realizing that it is the right house to display (see Fig. 2).

5. Temporal Reasoning

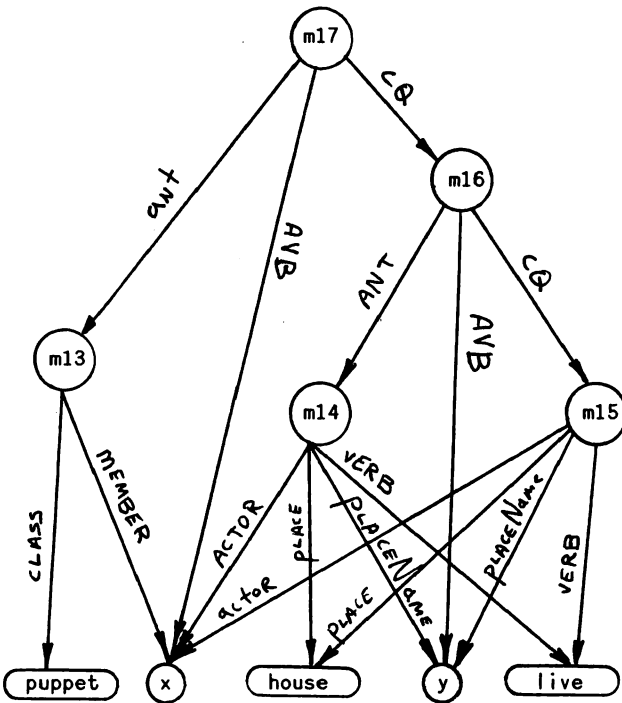
We can express the rules that all puppets are at work during work-hours and at home during off hours: "If it is a *freeTime*, then a puppet must be at home"; "If it is a *workTime*, then a puppet must be at work".

```

; For all x
;   if x is a puppet
;   then if currentState is workTime
;   then x is at his workplace.

(build avb $x
  ant (build member *x
        class puppet)
  cq (build ant (build
                currentState workTime)
      cq (build
          verb currentlyPresent
          actor *x
          place workplace)))

```



In the above rule "avb" arc stands for "all-variables-bound" and represents a universal quantifier. SNePS uses its rules in both forward and backward reasoning. In this rule, an English description is given first, (lines starting with semicolons) an actual SNePS User Language code follows and then a semantic network which is actually built. In this semantic network rule, node **m17** represents the entire rule. If the antecedent of the rule has matches in the network (node **m13**), then a consequent of the rule (node **m16**) is executed. But the network under node **m16** is also a rule.

The system may not know what a home (or a workplace) of **x** is. It just assumes that an actor is at some workplace or home. Then, it will have to deduce what the particular home or workplace of **x** is. (Two SNePS rules have been created to make this inferences.) To decide whether it is a workTime or a freeTime, the next set of rules is built:

```
(build avb ($day $hour $minute)
  ant (build name: isItWorkTime
    dayOfTheWeek *day
    hour *hour
    minute *minute)
  cq (build currentState workTime))
```

```
(build avb ($day $hour $minute)
  ant (build name: isItFreeTime
    dayOfTheWeek *day
    hour *hour
    minute *minute)
  cq (build currentState freeTime))
```

These two rules assert that it is a workTime or it is a freeTime if and only if a function node "isItWorkTime" or "isItFreeTime" succeeds. The system uses non-monotonic reasoning, so the state may change from workTime to freeTime. The system must not use old assertions about the time; instead, it must check the time again if it needs to. The system solves this problem by removing the temporal assertions each time, thus forcing the system to deduce them again.

We have tested our system by asking it different queries during different times of the day. During "freeTime" we ask the system to display the situation at the home of Ron (Fig. 3). During day time we ask the system to show the situation at a home of Bill (see Fig. 5). In the evening, we ask the system to display the workplace of Jane (Fig. 6).

When SNePS needs to display a set of objects, it creates a process which queries Graflisp about its capability to display those given objects. If Graflisp is capable of displaying the given set of objects, it collects, orders, orients, composes and renders that given set into an image. Then Graflisp passes a success message to the SNePS process, which consequently enables SNePS to deduce that the conceptual request is displayable. If the description which SNePS was provided with on a conceptual level can not be visualized in an image, Graflisp will be unable to find the corresponding objects or rules necessary to compose the scene and will pass failure back to the SNePS process.

Based on the information being passed from SNePS, the Graflisp module of the system is responsible for composing and rendering the image within the constraints of its view camera. The view camera determines the area of space to be viewed, the degree of perspective deformation, and the orientation of the image it will produce.

6. Scene Composition

After the system has inferred which objects are actually going to appear in the picture, it is the

responsibility of the graphics module to arrange these objects into a coherent scene and produce the image. To do this, first it must gather all of the conceptual constraints to be placed on the image and extrapolate them into three dimensional environmental constraints. The system starts this process by calculating a bounding hull for each object involved; these hulls are used to speed up rough calculations and insure that no two solid objects will occupy the same space.

The graphics module (Graflisp) starts to build an object hierarchy for the scene. The algorithm that it uses is very similar to the way in which a photographer might shoot a picture of a table-top scene. After using the hulls to calculate how much space it will need, Graflisp creates a sufficiently large blue backdrop and a horizontal green surface to serve as the sky and grassy ground for the environment in which it will place the objects. It then places a simulated camera into that environment at what would be eye-level for a puppet, and adjusts the camera's settings accordingly (lens angle, focal point, f-stop, etc). Additionally, a light source is set up above and behind the camera to serve as the Puppet World sun.

Next, the graphics module utilizes production rules to implement the particular preferences of the user. For example, if the user prefers to see puppets to the left and buildings to the right in images, then the system will query its objects as to their object class and order them from left to right accordingly, relative to the camera. Given this left-right ordering, and allowing an equal amount of image space for each object, three dimensional constraint pyramids are extrapolated out from the camera's film plane into the scene to serve as constraints on each object's placement in the scene. Objects are then placed into the scene as near to the camera as is possible without the object's bounding hull violating the object's constraint pyramid. Once all of these constraints are used to determine the objects' placements, Graflisp completes structuring the scene hierarchy and uses a z-buffer algorithm to render the scene into an image as it would have been seen by the simulated camera.

7. Reasoning about the display

How does the inference engine know whether the abstract concepts have a physical counterpart, and if they do, whether the graphics module is capable of displaying them? In order to perform further reasoning about the picture, the system has to have a way to assume that some picture elements have been displayed. A solution is to attach special Lisp functions to our rules, which will query the graphics module about its success or failure. SNePS *function nodes* give a procedural attachment capability to

the otherwise declarative style of SNePS programming. To "prove" a function node, the system must call the function which is associated with it. This is the actual SNePS-Graflisp interface. Here is one example of such a display rule:

```

; For all x, y, and z
;   if x is working in a place z by the name y
;   then if process of drawing y succeeds
;   it must be the case that we displayed it.

(build
  avb ($x $y $z)
  ant (build verb work
    actor *x
    place *z
    placeName *y)
  cq (build
    ant (build name: showPicture
      placeName *y)
    cq (build action display
      description
        (build verb work
          actor *x
          place *z
          placeName *y)
        type *z)))

```

This rule allows the system to reason about a particular action: displaying the workplace of one of the puppets known to the system. The arc named "name:" is a special system predefined arc which points to a function node. A function node creates a process which calls a Lisp function with an argument bound to *y*. The process can either succeed or fail. If the above process succeeds, then the consequent rule is asserted.

If we had used this rule in a backward chaining, the system would have tried to prove that for some actor *x*, working somewhere at place *z*, with the "placeName" *y* the picture had been displayed. If none of the variables were bound, this would be similar to the query: "Show me all the places of work for all the puppets". If, on the other hand, some of the variables were bound, it would be a reference to some specific instance of *x*, *y* or *z*. For example, if *x* is bound to Ron and *y* and *z* are free, this would be equivalent to the query: "Show me the place in which Ron works". To prove this, we would have to prove that the entire nested rule is valid. This rule would create a SNePS process, which would call Graflisp. When Graflisp displays the picture, the function node succeeds and returns true, and then the final consequent would be asserted. An English description of the rule for displaying a puppet's living place is provided below.

```

; For all x, y, and z
;   if x is living in a place z by the name y
;   then if process of drawing y succeeds
;   it must be the case that we displayed it.

```

For example, if we ask the system to show the house where Ron lives, the White House will be displayed (see Fig. 4).

8. Conclusions

The developed system allows us to relate the meaning of a picture to its graphic representation. Its two major blocks facilitate the intelligent computer graphics system's needs for reasoning about the content of a picture, the picture composition and the image generation. By interlinking SNePS and Graflisp, we have been able to obtain images originating from abstract requests. We have introduced a simple environment consisting of several puppets living and working in several places (called Puppet's World). Using abstract concepts (such as a home or a workplace) our system has successfully deduced which objects *should* be displayed and has displayed them.

In the next phase of this research, we plan to incorporate default display rules of user preferences. To collect and generate such rules, we will use the system described in [3]. We plan to develop an interactive visual test generator and rule acquisition package which can be used to customize default display rules to the preferences of a particular user as well as to the preferences of different classes of users.

References

- [1] Gardner, Brian R., "GRAFLISP: A Graphics Package Design for Artificial Intelligence Applications", Masters Thesis, Department of Computer Science, Boston University, 1985.
- [2] Hołyński Marek, Brian R. Gardner and Rafail Ostrovsky, "Towards Intelligent Computer Graphics System", *Technical Report*, Boston University, 1986.
- [3] Hołyński Marek and Lewis, Elaine, "Effective Visual Representation of Computer Generated Images", *IEEE Proceedings, 5th Symposium on Small Computers in the Arts*, IEEE Computer Society Press, 1985.
- [4] Shapiro, Stuart, "The SNePS semantic network processing system", *Associative Networks*, N.V. Findler (ed.), Academic Press, pp179-203, 1979.

EXPERIENCES WITH USING PROLOG FOR GEOMETRY

Wm. Randolph Franklin

Computer Science Division, 543 Evans
University of California, Berkeley, CA, 94720, USA.
(Arpanet: wrf@Berkeley.EDU)

and

Rensselaer Polytechnic Institute
Troy, NY, 12180, USA
(wrf%RPI-MTS.Mailnet@MIT-Multics.ARPA)

Margaret Nichols

North American Philips Lighting Co., Bloomfield NJ, USA

Sumitro Samaddar

Peter Wu

Rensselaer Polytechnic Institute
Troy, NY, 12180, USA
(Peter_Wu%RPI-MTS.Mailnet@MIT-Multics.ARPA)

ABSTRACT

The Prolog language is a useful tool for geometric and graphics implementations because its primitives, such as unification, match the requirements of many geometric algorithms. We have implemented several problems in Prolog including a subset of the Graphics Kernel Standard, convex hull finding, planar graph traversal, recognizing groupings of objects, and boolean combinations of polygons using multiple precision rational numbers. Certain paradigms, or standard forms, of geometric programming in Prolog are becoming evident. They include applying a function to every element of a set, executing a procedure so long as a certain geometric pattern exists, and using unification to propagate a transitive function. Certain strengths and weaknesses of Prolog for these applications are now apparent.

RÉSUMÉ

Le langage Prolog est un outil très utile pour la conception de logiciels géométriques et graphiques. Ceci est dû au fait que ses primitives, comme par exemple l'unification, répondent bien aux exigences de nombreux algorithmes géométriques. Nous avons résolu en Prolog plusieurs problèmes dont la représentation d'un sous-ensemble de la norme graphique Kernel, la détermination d'enveloppes convexes, le traitement de graphes plans, la reconnaissance de familles d'objets et la réalisation de combinaisons booléennes de polygones utilisant des nombres rationnels à précision élevée. Certaines hypothèses ou formes standard de programmation deviennent évidentes en Prolog. Ceci est vrai entre autre pour l'application d'une fonction à tous les éléments d'un ensemble, l'exécution d'une procédure tant qu'un certain motif géométrique existe et l'utilisation de l'unification pour la propagation d'une fonction transitive. Certaines forces et faiblesses de Prolog vis à vis de ces applications sont maintenant apparentes.

KEYWORDS: Prolog, Geometry, Graphics Kernel Standard

INTRODUCTION

The fifth generation logic programming language Prolog[Clocks81a, Coelho80a], appears appropriate for research in geometry and graphics. Some examples of its use in architectural design are given in [Swinson82a, Swinson83a, Swinson83b]. Its use in CAD has been evaluated in [Gonzalez84a]. Constructing geometric objects from certain constraints is described in [Brüderlin85a]. Over the past two years, the authors of this present paper have implemented several geometric and graphic problems in Prolog using assorted machines. This paper describes the experiences, including some paradigms of programming that have appeared useful, and finally listing the advantages and disadvantages of Prolog that we have experienced.

Over the last two years we have implemented several graphics and geometric algorithms in Prolog, totally a few thousand lines of code, using four different Prolog interpreters on four different computers. The systems include:

Machine	Operating System	Prolog Version
IBM 3081	Michigan Terminal System	York (U.K.)
IBM 4341	CMS	Waterlog
Prime 750	Primos	Salford
VAX 780	Unix bsd 4.3	UNSW

This work was supported by the National Science Foundation under grant no. ECS-8351942, the Data Systems Division of the International Business Machines Corporation, and by the Rome Air Development Center under the postdoctoral development program via Syracuse University.

The implementations include:

- Graphics Kernel Standard subset
- Convex Hull
- Planar Graph Traversal
- Big Rational Numbers
- Polygon Intersection
- Organization Inference

They will now be described in detail.

IMPLEMENTATIONS

Graphics Kernel Standard Subset

This graphics addition to Prolog was implemented by Nichols [Nichols85a] on an IBM 4341 using Waterlog [Roberts84a], under the CMS operating system. This allowed us to draw lines and so on on the 3270 graphics CRT from a Prolog program. We implemented two classes of lines: *permanent* and *backtrackable*. If the Prolog procedure that drew a backtrackable line was backtracked over, then the line would be erased. This used a feature of the graphics package GSP.

The major problems were as follows. Waterlog, like most Prologs, lacks floating point numbers, and even four byte integers. (The latter was undocumented; large integers just didn't work.) However it has the powerful capability to be linked to programs in other languages such as Fortran. Thus we implemented a real number in Prolog as a data structure of the form `real(A,B)` where A and B are Prolog integers holding the upper and lower halfword, respectively, of the integer. The user never looks at A and B, but accesses the real numbers via procedures such as `addreal(X, Y, Z)` and `realtointeger(R, I)` that took real numbers in the stated form and did the obvious things.

Convex Hull

This Graham Scan algorithm was implemented by Wu [Franklin85a] on both the IBM 4341, and on the Prime in Salford Prolog [Salford84a]. The Salford system allows both real numbers and dynamic linking to Fortran routines. We also tested York Prolog [Spivey83a], which is written in Pascal. The York system has the advantage that it is portable to any machine that can compile a thousand line Pascal program that uses four byte integers. Unfortunately this did not include the official Pascal compiler available from Prime. (We have not evaluated third-party Pascal compilers for Prime computers.) We also tested York Prolog on an IBM 3081 running the Michigan Terminal System, but found the other computers' operating systems more flexible and cheaper to use.

The algorithm proceeds as follows, using a divide and conquer paradigm. Duplicate points are removed and then the set of points is split into a left and a right subset based on the points' X-coordinates. The convex hulls of these sets are found recursively. To merge the

two convex hulls, the top and bottom tangents or supporting lines are required. The first approximation to the top tangent is found by joining the top point of the left convex hull to the top point of the right one. Then if necessary these endpoints of the tangents are moved right and left until the tangent does not intersect the convex hulls (except at the endpoints). This algorithm takes time $T = \theta(N \log(N))$.

The Prolog code is about 200 lines including comments.

Boolean Combinations of Polygons

A program to perform operations such as intersection, union, and difference on two planar polygons was implemented by Franklin and Wu [Franklin85a] on the Prime and IBM 4341. The algorithm was by Franklin [Franklin85a]. Wu first implemented a package to perform arithmetic using rational numbers in multiple precision. Each number, in life a quotient of an integer numerator and denominator, is implemented as a list of the numerator and denominator. Each of them is a list of groups of the digits of the number. For example, 123456789/987654321 is represented as `[[56789, 1234], [54321, 9876]]`. Rational numbers are used to avoid roundoff errors, as part of an ongoing investigation into their utility in geometry and the map overlay problem in cartography [Franklin84a].

The big rational package was designed in several steps as follows. First, rational numbers were implemented. A rational number Q is stored as the expression N/D . This is upward compatible with integers since is, which knows nothing of rationals, thinks it is just an integer expression. This also means that the rational number prints normally without a separate print procedure. We implemented a new infix operator, `isr`, which operates on rationals just as `is` operates on integers. It also converts integers to rationals. Rational versions of all the integer arithmetic operators were also implemented.

Next, a big integer arithmetic package was implemented, along with a new infix operator `are` and big versions of all the operators. Each big integer is stored as a list of groups of digits. For 32 bit built-in integers, each group is 4 digits. Zero is stored as `[]`, one as `[1]`, 72 as `[72]`, 10001 as `[1, 1]`, 2180077 as `[80077, 21]`, minus one as `[-1]`, -123456 as `[-56, -1234]`, and so on.

Then these were combined into one package with the operator `isx`. Now we can say things like

`X isx ([3456,12] + 23) / [222,3].`

The big rational package was tested by calculating π from the following formula, whose simplicity over-rides its very slow convergence.

$$\pi = 2 \cdot \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdot \dots$$

The UNSW Prolog code to execute this is:

`pi([],[2]). % preset value: Pi = 2`

```

step :-
    pi(R,P),
    R1 are R+[1],
    R2 are R1 mod [2],
    P1 isx P*((R1+R2)/(R1-R2+[1])),
    retract(pi(R,P)),
    asserta(pi(R1,P1)),
    pb(R1),prin(' '),pxq(P1),nl,!.

```

go :- repeat,step,fail.

The polygon combination system uses an edge based boundary representation. Each polygon is considered a set of edges. Here are the actual data structures.

```

vert(vertex_name, x, y)
edge(edge_name, name_of_first_vertex,
      name_of_second_vertex)
edge_eqn(edge_name, a, b, c)
poly(polygon_name, edge_name, which_side)

```

The edge equation is of the form $ax + by + c = 0$. There is one poly fact for each edge of each polygon. Since a given edge may be used by more than one polygon, it is necessary to know which side of the edge is the inside of this particular polygon. Legal values are left and right.

With this data structure, special cases involving multiple edges all ending at the same vertex are not a problem; in fact, the algorithm never knows of their existence. This data structure also does not store any global topology, such as the number of connected components, and which are inside which other. This information, which could be calculated if needed, is in fact never necessary.

The first stage of the algorithm is basically a forward reasoning system. It searches for cases where two edges intersect. Whenever this is found, those two edges are deleted, and three or four new edges are created. There will be three new edges if one edge's endpoint falls on another edge. This includes the case where the two edges are collinear. This process continues until no edges intersect, except possibly at both their endpoints.

This process is a little more complicated than appears since we are modifying the list of edge facts as we are iterating through it. This is one of the areas where different versions of Prolog behave differently. One solution is as follows.

1. Handle deletions not by actually retracting the edge, but by asserting a `deleted(edge_name)` fact to record the information.
2. Initially consider all edges to be of *level 0*.
3. Compare all the edges pair by pair. Whenever an intersection is found between two edges that do not have an associated deleted fact, then
 - a) assert a deleted fact about both of them, and
 - b) create three or four new edges by asserting *level 1* edges.

4. Then compare all the level 1 edges against each other and against all the level 0 edges without deleted facts. If any intersect, assert new *level 2* edges and deleted facts about the intersecting edges.
5. Then compare all the level 2 edges against each other and against all the level 0 and 1 edges.
6. Repeat this until no new intersections are found.
7. Finally clean up the database.

The above procedure should be portable since it does not modify any particular fact as control is iterating through instances of that fact.

Next in the boolean combination, each edge is classified into one of six categories:

- an edge of polygon A that is inside polygon B,
- on A outside B,
- on B inside A,
- on B outside A,
- an edge that is on both polygons A and B, and both polygons are on the same side of it, and
- on both polygons, and they are on opposite sides of it.

Finally, a subset of the edges is selected depending on the particular result desired. For example, in a union, edges on either polygon that are outside the other polygon, plus edges on both polygons with both on the same side, are needed. Since this selection takes almost no time, all the boolean combinations are found at no extra cost. For example, see figure 1, where polygon A is *ABCD* and polygon B is *EFGHIJ*. After intersecting edges are cut, edges *AB* and *EF* are cut into *AB*, *EB*, and *BF*. *HI* is cut into *HC* and *CI*. When the resulting edges are classified, edge *AB* is on polygon A outside of B. Edge *EJ* is on B inside A. Edge *EB* is on both polygons A and B, and they are on the same side. In contrast, edge *CI* is on both polygons, but they are on opposite sides.

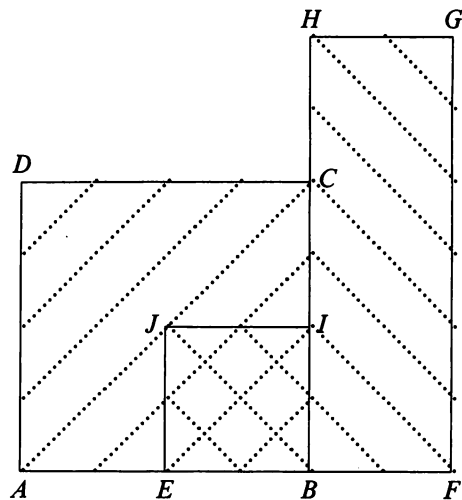


Figure 1: Combining Polygons *ABCD* and *EFGHIJ*

Planar Graph Traversal

At some point during an object space hidden surface algorithm [Franklin80a], we have the set of the visible edges and must join them to find the visible polygons. This requires a planar graph traversal, sometimes called a tessellation. For example, in figure 2,

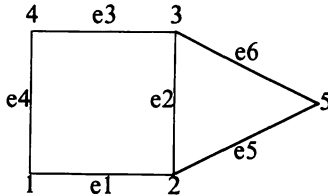


Figure 2: Finding the Faces of a Planar Graph

we are given the vertices and edges in the form

```
vert(vert-name, x-coord, y-coord)
edge(edge-name, first vertex, second vertex,
      angle)
```

for example

```
vert(v1, 0, 0)
edge(e1, v1, v2, 0)
```

The angle of the edge is supplied because of the difficulty of computing arctangents using only integers. The output is a set of facts of the form

```
polygon([v1, v2, v3, v4])
```

This was implemented in UNSW Prolog [Sammut83a] on a Vax.

Organization Inference

In this work, described in more detail in [Samaddar85a], we wish to infer which units of an army organization are present after seeing, via photoreconnaissance, an incomplete picture of the equipment they possess. The army organization, parts of which may be present in the photo, is described with Prolog facts such as the following.

```
child(Father, Son, Number)
```

This says that unit Father ideally contains Number of the subunit Son. For example, a parts of Soviet motorized rifle division might be defined thus:

```
child(motorized_rifle_division,btr_regiment,2).
child(motorized_rifle_division,bmp_regiment,1).
child(motorized_rifle_division,tank_regiment,1).
child(motorized_rifle_division,
      artillery_regiment,1).
child(bmp_regiment,bmp_battalion,3).
child(bmp_battalion,bmp_company,3).
child(bmp_company,bmp_platoon,3).
```

The equipment that each unit possesses is described by the following form of fact:

```
eqpmnt_overall(Unit, Ename, Number)
```

Unit is the name of the unit that owns the equipment, such as art_reg for an artillery regiment. Ename is the name of the equipment, such as sa-6 for an SA-6 anti-aircraft missile. Number is the maximum number of pieces of equipment that that unit can own. The fact for a particular unit includes only equipment that the unit owns directly, and not equipment owned by a subunit. Some sample facts are:

```
eqpmnt_overall(art_reg, sa_6, 20).
eqpmnt_overall(mr_div, amphi_brdm, 48).
```

Then facts defining what equipment has been recognized are stated as follows:

```
equipment(Name, Number)
```

For example,

```
equipment(sa_7, 7).
equipment(rpg_7, 23).
```

Given this information, the inference engine reports that

Based on that, my first guess about the unit present, and the remaining equipment associated with it, is:

```
Remaining = [[arm_per_car_btr, 38],
             [mortar_120mm_1943, 6]]
Unit = mot_rif_btl_n_btr
```

This inference engine is designed to be part of a larger blackboard format system where a low level image interpretation and geometry engine makes a first guess about the objects present and passes the information up to this unit. The output of this unit can be used to bias the prior probabilities of the geometry system as it continues to look.

This system is robust since it automatically handles the cases of the unit on the ground being under strength, and the image interpretation system not finding everything.

STRENGTHS AND WEAKNESSES OF PROLOG

Certain advantages and disadvantages of Prolog for graphics and geometric applications are becoming evident from these implementations.

Advantages Of Prolog

- Prolog has same high level advantages of Lisp, as the equivalence of code and data and dynamic data allocation.
- There are the specific advantages of Prolog. Unification makes determining graph connectivity a primitive operation and in general is useful for propagating transitive properties such as graph connectivity which occur frequently. This is a counterexample to the proposition that, "Unification is what you do when you don't know what you are doing".

- The pattern matching fits with the form of expression of many algorithms. For example, our polygon combination algorithm proceeds as follows. Whenever the pattern of two edges intersecting, or one edge ends on the interior of another edge, occurs, then retract those edges and assert new smaller edges. When this pattern no longer exists, then we have a superset of the edges in the output polygon.
- Although many of the above features could be implemented in any language that is Turing equivalent, Prolog is somewhat standard so that different researchers can understand and use each others' extensions.

Disadvantages Of Prolog

However, there are some problems with using Prolog for geometry.

- There are software engineering problems with using Prolog for a large project because of its lack of nesting in the program and databases.
- Many geometry algorithms are more natural to a forward reasoning system than a backward reasoning system. That is, we are more likely to want the output from some given input than the reverse.
- The natural way of expressing pattern matching algorithms requires us to modify a database as we are searching through it. Thus in polygon overlay, whenever we find the pattern of two edges crossing, we retract them and assert four new edges. Backtracking and redoing a database that we are modifying does not work on all Prologs.
- Prolog does not support coroutines, which are a natural way to express many algorithms.
- In general Prolog is completely unstandardized around the fringes as some tests of cuts in [Moss85a] show.

PARADIGMS OF PROGRAMMING

Certain techniques have proven to be generally useful in our implementations, and may be useful to others also. They include the following paradigms.

Set Based Algorithms

Many algorithms such as polyhedron intersection and hidden surface algorithms, Franklin [Franklin82a, Franklin80a], are the alternation of two types of steps:

- Applying function to every element of a set, and
- Combining all the elements having a common key.

This is clearly easy in Prolog.

Pattern Matching

The second paradigm uses pattern matching to propagate certain properties. For example, in the planar graph traversal algorithm, the edges around each vertex are found and sorted by the angle at which they

leave it. Then the edges around each vertex are paired to form *corners*. These corners can be considered to be fragments of the output polygons. Whenever two fragments exist such that the last edge of one is the same as the first edge of another, then these two fragments are retracted and a single longer fragment asserted. When such a pattern no longer exists, then we have the output polygons.

Unification

Frequently we wish to determine the closure of some transitive property, such as when we are given a set of graph edges $\text{edge}(u, v)$, and wish to determine the connected components. We have implemented the following short algorithm that uses unification and the set processing paradigm.

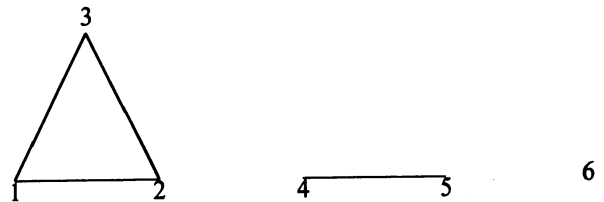


Figure 3: Determining Graph Connectivity

- Create a property list (*plist*) with one record per vertex, and the property of each vertex a free variable. For example in figure 3 we would have $[[1, _], [2, _], [3, _], [4, _], [5, _], [6, _]]$.
- Process the set of edges and for each edge unify the free variable properties of the endpoints. After this we will have $[[1, _1], [2, _1], [3, _1], [4, _2], [5, _2], [6, _3]]$ with one unique free variable per graph component.
- Bind a name identifying each component to the free variables in the list to give something like $[[1, \text{first}], [2, \text{first}], [3, \text{first}], [4, \text{second}], [5, \text{second}], [6, \text{third}]]$.

A longer example of a simple hidden surface algorithm would go as follows.

- Wherever the pattern of two edges' projections' intersecting occurs, split the edges into four smaller edge segments.
- For each edge segment find the set of faces hiding its midpoint. If it is empty then the edge segment is visible. Draw them.
- Use a planar graph traversal algorithm such as described above to link the visible edges into polygons.
- For each polygon, find a point inside it and then find the set of faces whose projections contain the projection of that point. Find the closest such face; the polygon came from it. Color the polygon accordingly.

This illustrates all of the paradigms operating together.

SUMMARY

Although not perfect, Prolog is a powerful tool for expressing graphical and geometry algorithms in a concise and natural format. This allows larger problems to be solved in a given time, and raises the size of the largest problem that it is feasible to solve.

REFERENCES

- Brüderlin85a.
Beat Brüderlin, "Using Prolog for Constructing Geometric Objects Defined by Constraints," *Eurocal 85, Conference Proceedings*, Linz, Austria, 1985. Institut für Informatik, ETH Zürich, CH-8092, Zürich, Switzerland
- Clocks81a.
W.F. Clocksin and C.S. Mellish, *Programming In Prolog*, Springer-Verlag, New York, 1981.
- Coelho80a.
H. Coelho, J.C. Cotta, and L.M. Pereira, *How to Solve it With Prolog, 2nd edition*, Ministerio da Habitacao e Obras Publicas, Laboratorio Nacional de Engenharia Civil, Lisboa, 1980.
- Franklin80a.
Wm. Randolph Franklin, "A Linear Time Exact Hidden Surface Algorithm," *ACM Computer Graphics*, vol. 14, no. 3, pp. 117-123, July 1980. Proceedings of SIGGRAPH'80
- Franklin82a.
Wm. Randolph Franklin, "Efficient Polyhedron Intersection and Union," *Proc. Graphics Interface'82*, pp. 73-80, Toronto, 19-21 May 1982.
- Franklin84a.
Wm. Randolph Franklin, "Cartographic Errors Symptomatic of Underlying Algebra Problems," *Proc. International Symposium on Spatial Data Handling*, vol. 1, pp. 190-208, Zürich, Switzerland, 20-24 August 1984.
- Franklin85a.
Wm. Randolph Franklin and Peter Y.F. Wu, *Convex Hull and Polygon Intersection Implemented in Prolog*, Rensselaer Polytechnic Institute, Troy, NY, July 1985.
- Gonzalez84a.
J.C. Gonzalez, M.H. Williams, and I.E. Aitchison, "Evaluation of the Effectiveness of Prolog for a CAD Application," *IEEE Computer Graphics and Applications*, pp. 67-75, March 1984.
- Moss85a.
Chris Moss and Earl Fogel, *Tests to Distinguish Various Implementations of Cut in Prolog*, Imperial College and Logicware Inc., June 1985. Reported on Usenet in Net.lang.Prolog, message-id <1742@utecfa.UUCP>.
- Nichols85a.
Margaret Nichols, *The Graphic Kernel System in Prolog*, ECSE Dept., Rensselaer Polytechnic Institute, Masters Thesis, Troy, NY, August 1985.
- Roberts84a.
Grant Roberts, *Waterloo Core Prolog Users Manual (version 1.5)*, Intralogic Inc., Waterloo, Ont, Canada, 1984.
- Salford84a.
University of Salford, *LISP/PROLOG Reference Manual*, March 1984.
- Samaddar85a.
Sumitro Samaddar, *An Expert System for Photo Interpretation*, ECSE Dept., Rensselaer Polytechnic Institute, Masters Thesis, Troy, NY, August 1985.
- Samm83a.
Claude Sammut, *UNSW Prolog User Manual*, University of New South Wales (Australia), 1983.
- Spivey83a.
J. M. Spivey, *University of York Portable Prolog System (Release 1) User's Guide*, York, U.K., March 1983.
- Swinson82a.
P.S.G. Swinson, "Logic Programming: A Computing Tool for the Architect of the Future," *Computer Aided Design*, vol. 14, no. 2, pp. 97-104, March 1982.
- Swinson83b.
P.S.G. Swinson, "Prolog: A Prelude to a New Generation of CAAD," *Computer Aided Design*, vol. 15, no. 6, pp. 335-343, November 1983.
- Swinson83a.
P.S.G. Swinson, F.C.N. Periera, and A. Bijl, "A Fact Dependency System for the Logic Programmer," *Computer Aided Design*, vol. 15, no. 4, pp. 235-243, July 1983.

THE INFERENCE MACHINE LABORATORY: GRAPHIC TOOLS FOR KNOWLEDGE MANAGEMENT

J. W. Lewis, Ph.D.

Artificial Intelligence Department
Martin Marietta Laboratories
1450 South Rolling Road
Baltimore, MD 21227

ABSTRACT

The Inference Machine Laboratory is a collection of experiments in applying graphic interfaces to various types of knowledge bases. Each experiment involves a canonical representation, invertible transformations into multiple representations, and multiple directly manipulable views of those representations. Initial experiments include RULE*CALC (simple production rules), HAPStation (OPS5-like production rules), RFIX (diagnostics), and TIMLS (frames in PROLOG).

KEYWORDS: expert systems, intelligent interfaces, knowledge acquisition, direct manipulation

INTRODUCTION

The major barrier to successful expert systems development continues to be the acquisition, review, restructuring, and long-term maintenance of large knowledge bases involving complex relationships [1]. Meaningful military and industrial expert systems are expected to require as many as 10,000 "rules" [2], domain coverage of better than 95 %, and an error rate of less than 0.1 %. To achieve these performance figures -- perhaps one to two orders of magnitude beyond the current state of the art -- the next generation of knowledge management tools must enable each individual involved in designing, building, using, and maintaining knowledge bases to view, understand, and manipulate their contents in an intuitive manner.

For simple interactive systems, adequate tools and techniques are already available. The direct manipulation of icons has already lead to successful icon-based interfaces for commercial systems such as the XEROX Star and the Apple Macintosh [3]. Research activities, such as those in the MIT Media Graphics Laboratory, have shown impressive capabilities for text and video interfaces [4]. More recently, tools such as UNITS [5] and GEN-X [6] have provided effective interfaces to knowledge bases in expert systems.

THE INFERENCE MACHINE LABORATORY

The Inference Machine Laboratory (IML) addresses these performance goals for expert systems by providing each knowledge-base user with a tailored set of directly manipulable views of the knowledge base and by maintaining consistency among the various views. Over the last two years, a family of successively more complex knowledge management systems has been constructed. The early systems have involved simple production rule knowledge bases, and the later systems are based on predicate calculus representations.

Physically, the laboratory consists of two VAX computers, several LISP machines, a color monitor, a color video projector, several mice, a foot-mouse, a DECTALK voice generator, and a Polhemus 3-D graphics pointing device. The half-dozen individuals interacting with the knowledge base are grouped around a small table in front of the projection screen. They are able to select views, make queries against the knowledge base, and modify the knowledge base using the interactive graphics devices.

All of the software systems in the laboratory fit into a common framework (Fig. 1), which supports various kinds of graphic input/output, logic-based knowledge representations, and natural language input/output. In the IML, each system user works with a particular set of windows on the knowledge base. These windows are defined by

- o Virtual cameras - to generate shaded images, schematics, tables, graphs, trees, and other diagrams
- o Views - to specify the particular image generated by defining the location of the user in the knowledge base,
- o Filters - to determine the granularity, or level of detail, in a particular view.

Once a view is presented, the user can modify the knowledge base by pointing at particular elements of the view and indicating changes.

RULE*CALC

The simplest and earliest project in the laboratory is RULE*CALC, a VISI-CALC-style development environment for EMYCIN-class production rules with uncertainty [7]. The rules are laid out in a spreadsheet-like tableau.

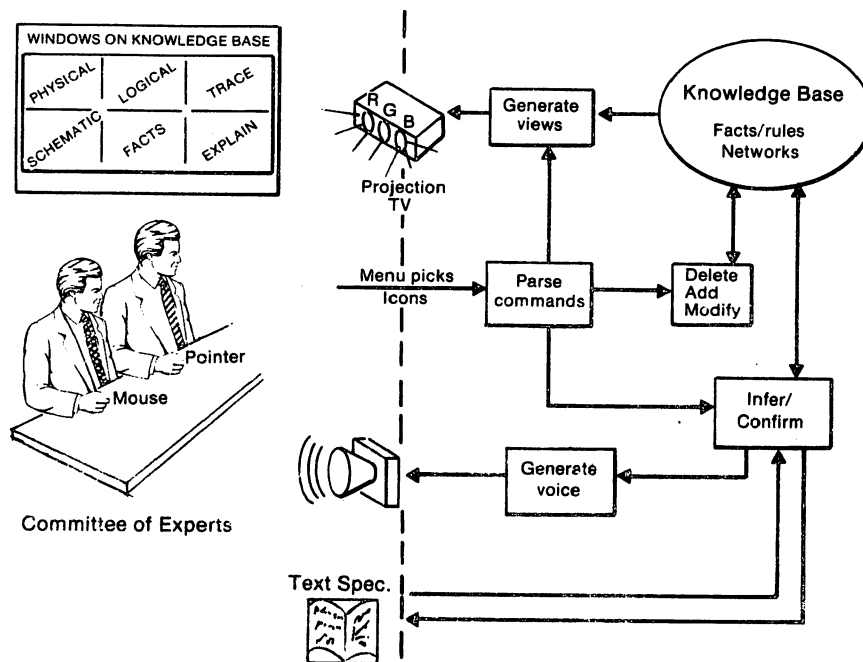


Figure 1. The Inference Machine Laboratory framework

(See Fig. 2 for a few of the rules in a 300-rule system for FAA radar trouble-shooting.) Each row in the tableau corresponds to a fact in the rules, and each column represents one rule. Rules are defined by entering a symbol in the appropriate column, which indicates how that fact is involved in the particular rule. A blank indicates that the fact is not involved in the rule at all; an = or ~ symbol indicates that the fact is a positive or negative clause in the lefthand side of the rule, and an :-T or :-F symbol indicates that the fact is asserted or negated when that particular rule fires. The user modifies the table by pointing to the appropriate entry with a mouse and hitting function keys for setting symbols in the table, cutting/pasting rules (rows), cutting/pasting facts (columns), and looking at different windows on the set of rules.

Each fact defines a simple object which could have one or more of three associated action procedures (methods): on-demand (triggered when a value is requested), on-true (triggered when the fact is asserted true), and on-false (triggered when the fact is asserted false). If the system is run from a terminal, the methods type out messages on the screen and elicit responses from the keyboard. If the system is run by calling in on a telephone, the methods generate a voice over the phone and elicit responses from the telephone keypad.

When the system executes, the state of the inference engine and the resulting dialog can be displayed in a pair of windows in the rule-debugging screen. One window shows the interactive dialog and the other displays the rule stack along with the facts involved in those rules. Altogether, in RULE*CALC there are five fixed views: the rule-edit tableau, fact-edit tableau, method definition, interactive dialog, and rule-debugging.

HAPStation

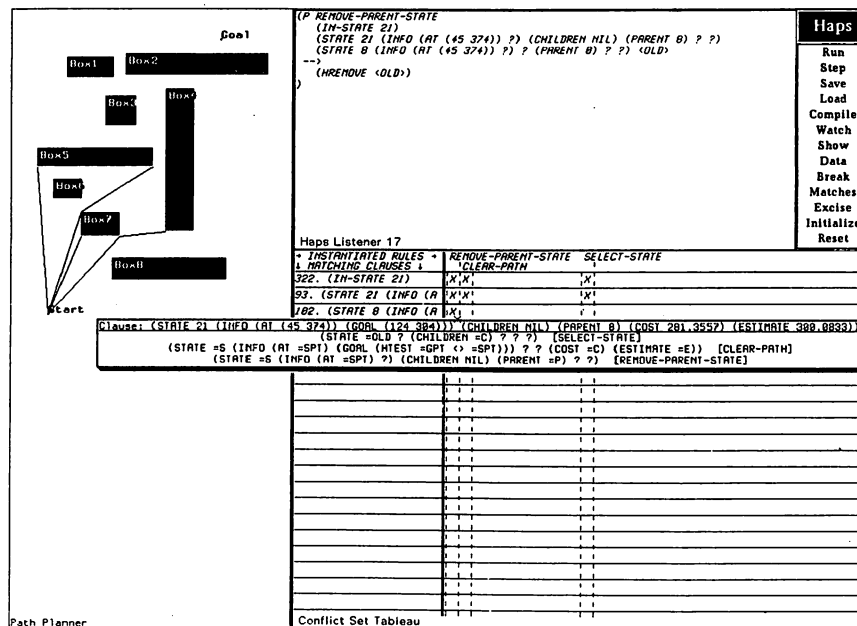
The second system is HAPStation, a somewhat more complex inference engine with a simpler interactive interface. HAPStation runs on a SYMBOLICS LISP workstation in COMMON LISP (Fig. 3). HAPS is a forward-chaining, production-rule-based language similar in style to OPS4, GRAPES, and OPS83 [8,9]. Like other forward-chaining production rule systems, HAPS is composed of two memories -- a working memory (WM) and a production rule memory (PM) -- with an accompanying interpreter. The working memory elements are composed of a sequence of terms in parentheses, e.g., "(this is a working memory element)." The production rules are composed of a lefthand side (also called the LHS, antecedent, or IF-part) and a righthand side (also called the RHS, consequent, or THEN-part). The LHS of each rule is composed of a sequence of positive and negative clauses, which are in turn composed of the patterns to be matched against the WM. The RHS of each rule is composed of a sequence of action terms involving making and removing working elements, computing expressions, and input/output.

The interpreter cycles through a recognize-act cycle in which it first searches for all working memory elements that match the clauses in the LHS of the rules. The resulting set of rules is called the conflict set (CS), and the rule that fires is selected from the CS by a programmable conflict-resolution strategy. When the rule fires, the terms in its RHS are executed in sequence. This cycle continues indefinitely until the conflict set is empty or a rule executes a HALT action.

The interactive interface is composed of a set of windows on the memories in the production rule interpreter.

The central window (or HAPS window) contains the normal sequential dialog with the interpreter. The HAPSedit window enables the user to modify the rules through the ZMACS "smart" editor on the LISP machines. Other windows display the WM, the CS, and the possible matches with the LHS of a rule. Many of the elements of the screen are "mousable" so that the user can call up a pop-up menu of actions (run, stop, etc.) and a menu of the rule names against which to apply those actions.

The third project area is IDS (Integrated Diagnostic Systems). This project is focussed on a specific application: isolating and addressing faults in complex systems. In particular, the project addresses intermittents, multiple faults, consequent faults, design errors, unusual operating modes, and other failures that challenge teams of experts. The IDS has two interfaces, one that serves the domain



Vision Interface '86

expert in building or applying the knowledge base and another that serves the end user during the troubleshooting process. The expert interface shows multiple views of the device and of the status of the inference engine working against it. For example, the RFIX system in Fig. 4 shows six distinct windows on the robot being diagnosed, the robot's schematic diagram, the dialog, and the underlying inference engine.

The physical window is generated by a Giroud shading algorithm from a faceted surface model of the robot. The drive motors are shown through the translucent skin of the robot. The schematic view shows a block diagram of the LSI11 CPU and of the analog control system. The other windows show the current action, current question, and the current rule stack. In each window, the status of each of the elements of the robot control system is indicated by a color: blue to indicate state unknown but presumed good, green to indicate known good, and red to indicate a known failure. Eventually, all elements of the display will be independently mousable, enabling the user to call up parts descriptions, to explain the detailed state of the indicated object, and to request that tests be applied to the object.

PROLOG QUERY TABLE

The final project area is TIMLS (The Inference Machine Laboratory System). The focus of TIMLS is building and maintaining situation assessment or planning knowledge bases for autonomous vehicles. The current knowledge base is a complex PROLOG-based frame system that implements simple property inheritance and temporal logic using the method of temporal arguments [10]. The model scenario was drawn from the September 1943 British X-craft midget submarine attack on the German battleship Tirpitz [11]. All events in that attack are captured in a PROLOG data base. The data base includes more than one hundred relations of the form

```
break_clear (X-craft, Barrier, Time, Flags).
come_out_to (X-craft, Object, Time, Flags).
come_out_from (X-craft, Enclosure, Time, Flags).
dive_into (X-craft, Enclosure, Time, Flags) . . .
```

As in the other projects, the interface provides multiple interactive views of the knowledge base. Two of the windows are color-map views of the Kaa Fjord at different levels of detail. In those windows, the paths and positions

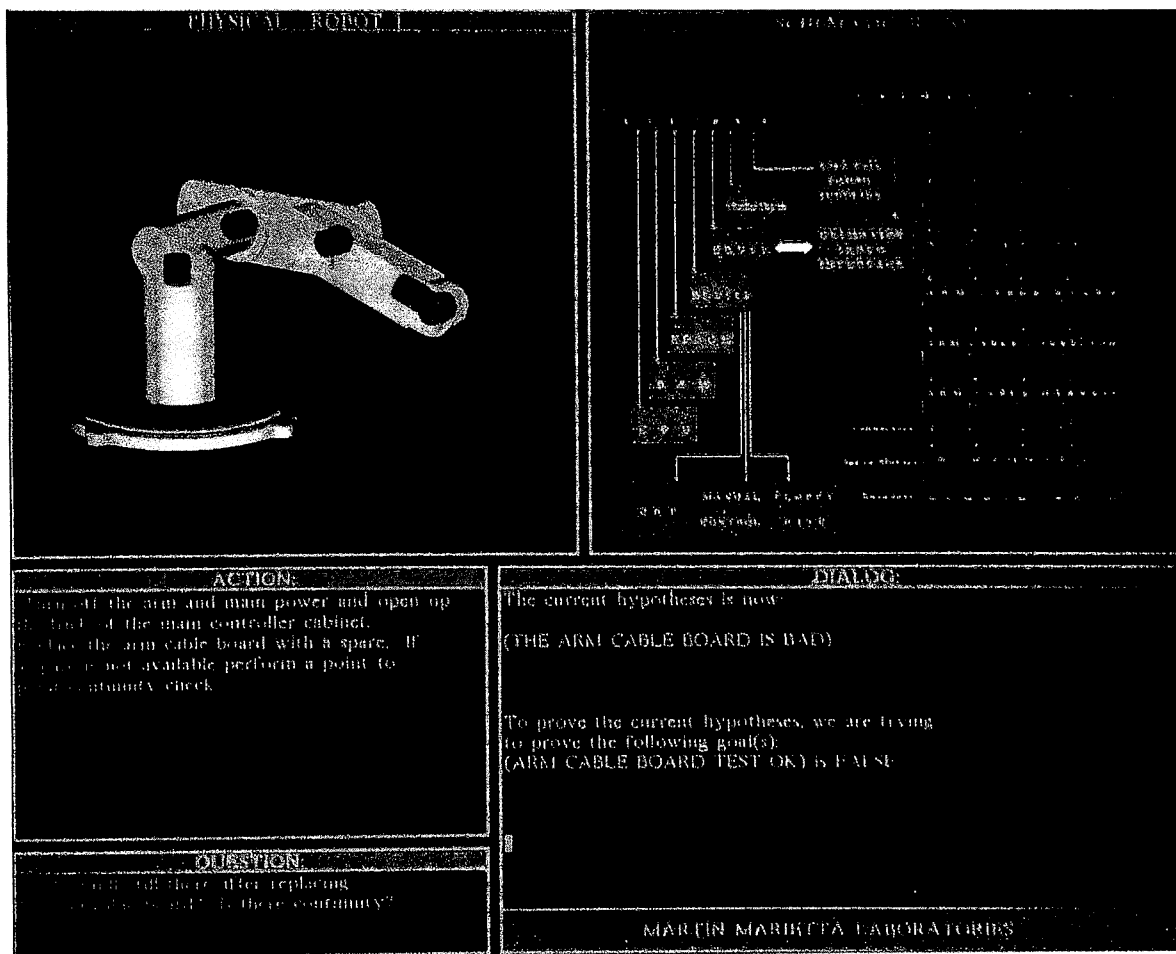


Figure 4. RFIX: The Robot Troubleshooting System

of the German and British ships are shown relative to obstacles such as submarine nets and buoys. The other windows contain the raw PROLOG, an English-language description of the weather generated from case frames, and the PROLOG query table (PQT).

Like RULE*CALC, the PQT follows the VISI-CALC paradigm. It is a close relative of the Query By Example (QBE) System [12] and PROLOG implementations of it [13]. The recently implemented PQT consists of two separate windows (Fig. 5). The upper window shows the current constraints on subsequent queries. Each constraint involves fixed values for one or more of the binary relations defined on the objects in the data base by the frame system. For the example given, the constraints are that the time is 9/22/1943 and that some number of X craft are in the Kaa Fjord.

SITUATION REPORT

Partly cloudy sky today will cover the upper section of Norway and showers will occur. The winds will be from the NW at 15 to 20 knots. The temperature will reach 35 to 40 degrees and visibility will be moderate to good.

PROLOG QUERY TABLE

Constraints...			
TIME	ISA	LOCATION	
X ON 9/22/1943	X-CRAFT	KAA-FJORD	
Query/Response...			
OBJECT	SINK	LOCATION	TIME
X-6	UNDERWATER	UNDER(STARBOARD(BOW(TIRPITZ)))	7:30 ON 9/22/1943
X-7	UNDERWATER	UNDER(BATTLE_ PRACTICE_ TARGET 1)	8:30 ON 9/22/1943

Figure 5. Two of the TIMLS windows

The header of the second window is the specific query on the data base. Each header element is the name of other relations defined by the frame system. The body of the table is the response to the constrained query in the form of a list of n-tuples. The other windows echo that query with path(s) on the map or a new natural-language weather report. Many of the elements in the PQT are mousable so that new relations (columns) and constraints (upper window) can be defined, or old relations and constraints removed.

ACKNOWLEDGMENTS

The key implementor for RULE*CALC has been J. Thorp, and H. Mayerfeld was responsible for the knowledge engineering on the ASR8 radar application. The HAPStation interface was coded by J. Sanborn as part of a collaboration with D. Marshall of Denver Aerospace. The development team for the RFIX interface included J. Mills (integration), C. Phillips (shaded graphics), and T. Burzio (diagrams). The TIMLS work was done by B. Haugh (logic), Y. Sekine (natural language), S. Barash (PQT), and B. Kobler (graphic interface).

REFERENCES

- [1] E. A. Feigenbaum, *Knowledge Engineering for the 1980's*. Technical report, Computer Science Department, Stanford University, 1982.
- [2] Defense Advanced Research Projects Agency (DARPA), *Strategic Computing New-Generation Computing Technology: A Strategic Plan for Its Development and Application to Critical Problems in Defense*. Oct. 28, 1983.
- [3] B. Shneiderman, "Direct Manipulation: a step beyond programming languages," *IEEE Computer* 8 (16), pp. 57-68.
- [4] R.L. Currier, "Interactive videodisc learning systems," *High Technology*, Nov. 1983, pp. 51-59.
- [5] D.A. Waterman, *A Guide to Building Expert Systems*. Addison-Wesley, 1985, p. 350
- [6] *Ibid.*, pp. 342, 359.
- [7] J.W. Lewis, J.R. Thorp, and H.M. Mayerfeld, *The Rule*Calc Manual*. Martin Marietta Laboratories report MML TR85-6, February 1985.
- [8] R. Sauers, "On the requirements of future expert systems," *IJCAI '83*, pp. 740-743.
- [9] D. Marshall and J. Sanborn, *The HAPS Users Manual*. Martin Marietta Laboratories Technical Report, 1985.
- [10] J. Allen, "Towards a general theory of action and time," *Artificial Intelligence* 24, pp. 123-154, 1984.
- [11] G. Frere-Cock, *The Attacks on the Tirpitz*. Naval Institute Press, 1973.
- [12] IBM Corporation, *Query-by-Example Terminal User's Guide*. IBM Form No. SH20-2078.
- [13] J.C. Neves, S.O. Anderson, and M.H. Williams, "A PROLOG implementation of query-by-example," in *Proc. 7th International Computing Symposium*, Nuremberg, Germany, pp. 318-332.

PORTRAY – AN IMAGE SYNTHESIS SYSTEM

Darwyn R. Peachey

Department of Computational Science
University of Saskatchewan
Saskatoon, Canada

ABSTRACT

PORTRAY is an image synthesis system which uses ray tracing to produce realistic images of three-dimensional scenes. Scenes are described to PORTRAY in a high-level description language. The basic geometric modelling technique is constructive solid geometry using primitive solids bounded by planes and quadrics. A variety of optical characteristics and phenomena may be specified. The scene description language allows the user to define object *classes* which may be used as if they were built-in primitives. PORTRAY uses a number of techniques, including a novel technique exploiting bounding volume coherence, to improve its ray tracing performance. PORTRAY is supported by an array of image manipulation tools which share a common image storage format.

KEYWORDS: bounding volume coherence, constructive solid geometry, illumination models, image synthesis, ray tracing.

1. Introduction

PORTRAY is an image synthesis system which generates realistic pictures of three-dimensional scenes. Scenes are described to PORTRAY using a high-level scene description language (SDL). The scene description is processed by a *scene compiler* called "PRCOMP". PRCOMP produces a file which describes the scene in a lower-level language. This intermediate file is read by the rendering program, simply called "PORTRAY", which uses ray tracing to produce an image of the scene. Figure 1 illustrates the structure of the system.

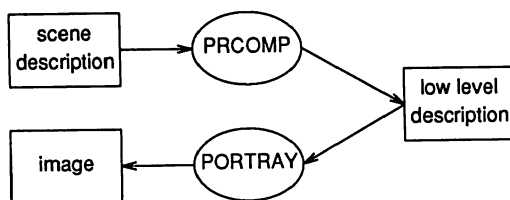


Figure 1

The PORTRAY image synthesis system is implemented in the C programming language, and runs under UNIX on VAX, Sun, and Pyramid computers.

This paper describes the geometric and optical modelling techniques used in PORTRAY, the scene description language, the processes of scene compilation and rendering, and the image format and image manipulation tools used with PORTRAY.

2. Scene Description Language

The SDL is a critical part of the PORTRAY system, since it determines the ease with which a user of the system can model the objects he wants in a given scene. With a particular SDL, some scenes may be impossible to describe, and many more scenes may be impractical to describe. The PORTRAY SDL incorporates a powerful geometric modelling technique and a variety of optical modelling techniques to give the user a large amount of descriptive power [1].

2.1 Geometric Modelling

The shapes of objects are described to PORTRAY by means of constructive solid geometry (CSG). CSG descriptions are expressions involving Boolean combinations of primitive solids. PORTRAY uses spheres, cones, cylinders, and cubes as primitive solids. These primitives have quadric and planar surfaces which make the problem of determining the intersection between a ray and the primitive quite simple. Primitives may be moved to arbitrary locations and rotated and scaled as desired. Unequal scaling may be used, for example, to turn a sphere into an ellipsoid, or a cube into an arbitrary rectangular block.

Primitive solids are combined using *regularized set operators* [2], namely union, intersection, and subtraction. The combination of two solids is guaranteed to be another well-defined solid. Any two CSG expressions can be combined using any of the three operators to obtain another CSG expression. Quite elaborate objects may easily be constructed in this way (see Figure 2).

The use of solid modelling instead of surface modelling entails some additional cost. To ray trace a solid consisting of a combination of primitive solids, we must first find the intersections between a given ray and each of the primitive solids. The resulting lists of intersections are then merged in a

way which depends on the semantics of the CSG operator. In general, we must find *all* intersections between a ray and the object, even though we only use the intersection nearest to the eye in most cases. This is because more distant intersections may affect the outcome of a merge. Ray tracing of surface models usually only requires that we find the nearest intersection between the ray and each surface. However, the additional cost of ray tracing solid models is justified by the geometric modelling power and simplicity of CSG. Solid modelling is also advantageous for simulating optical phenomena which involve light passing through the body of a solid.

Although the three CSG operators are inherently binary operators which take exactly two operand expressions, the PORTRAY SDL allows the use of "m-ary" (associative) union and intersection operations as a notational convenience. This reduces the need for the user to fully parenthesize complex descriptions to explicitly indicate the operands of every operator.

There is nothing sacred about the particular set of four primitive solids which are used by PORTRAY. In fact, the programs are designed so that a new primitive can be added simply by programming routines to find intersections, normal vectors, and texture coordinates for the new primitive, and by adding a line to a single table which links the new routines into the rest of the system. In future, primitives may be added to PORTRAY to more easily describe irregular, natural objects. CSG with simple quadric primitives is an effective means of describing most man-made objects, but the great complexity of natural objects would be modelled better by primitives such as fractal surfaces.

2.2 Optical Modelling

Shape is only one aspect of an object which must be modelled by an image synthesis system. We use the term "optical modelling" to refer to the other attributes of an object (color, texture, reflectance, transmittance, etc.) which determine how a ray of light interacts with the object. PORTRAY provides a wide variety of optical modelling techniques.

One of the most basic optical characteristics is color. The PORTRAY user may specify colors using an English-like scheme which is a simplified version of the color naming system (CNS) described in [3], or more precisely in terms of hue, saturation, and value. Both the CNS and HSV color models are generally believed to be easier for people to use than the RGB color model which PORTRAY uses for its internal computations.

The "shininess" attribute is used to specify the reflectance of a surface. The user may specify shininess in a pseudo-English fashion, using the keywords SHINIEST, SHINIER, SHINY, DULL, DULLER, and DULLEST. Alternatively, the reflectance of a surface may be specified numerically. The "smoothness" attribute controls the size of specular highlights which appear on a surface. A shiny smooth surface has smaller, sharper highlights than a shiny rough surface. The optical model simulates smoothness variations by controlling the parameter m in the Beckman function that determines the directional distribution of surface microfacets in the Cook-Torrance reflection model [4]. Extremely smooth

surfaces (specified as SMOOTHEST by the user) have ray traced reflections and transmissions.

Transmission and refraction of light through objects is controlled by the "transparency" attribute. The user may specify the index of refraction and a light scattering factor for each transparent object. The light scattering factor is expressed as a distance which a light ray would have to travel through the material in order to be reduced to one-half its original brightness. The relative contribution of the reflected and transmitted rays at an interface between transparent materials is determined using the Fresnel equations of physical optics. Figure 3 is an image of a scene containing a wine glass, which illustrates the use of reflection and refraction. If the index of refraction is specified as FAKE, PORTRAY allows rays striking the surface to be transmitted straight through without refraction.

The "pure" attribute is used to distinguish between composite materials such as plastics, where the color of highlights depends only on the color of incident light, and pure materials such as metals, where the color of highlights is influenced by the body color of the material.

The "paint" attribute is used to specify a variety of texturing techniques. For each texture, the user specifies a built-in texture function and a texture color. PORTRAY interpolates between the normal object color and the paint color according to the texture function. Some of the texture functions make use of disk files of textural information. In such cases, the user specifies the texture file by name. Texture files are stored in the same format as the images produced by PORTRAY (see section 5) and PORTRAY may use several texture files during the rendering of a single scene. The scene depicted in Figure 4 makes use of nine texture functions and seven different texture files.

PORTRAY has been used to experiment with a new texturing technique called "solid texturing" [5]. Solid texture functions proved to be easy to add to the library of built-in texture functions. The left and right spindles in Figure 5 show the application of two of these solid texture functions.

2.3 SDL Statements

A scene description in PORTRAY SDL consists of a sequence of *statements*. There are several types of statements, which are described in the following paragraphs.

CAMERA, TARGET, and FOCAL LENGTH statements specify the position, orientation, and focal length of the camera which is simulated by ray tracing. Since 35 mm cameras are popular and familiar, the simulated camera is designed so that the focal lengths of the lenses used with a 35 mm camera can be used in the FOCAL LENGTH statement to obtain similar effects.

OBJECT statements specify the CSG expressions and optical attributes which describe the objects in the scene.

LIGHT, AMBIENT, and BACKGROUND statements specify the intensity, color, position, and type of light sources, and the color of the infinitely large background sphere which surrounds the scene. A direct light source may be specified as a point source, a beam of parallel rays from a given direction, or a focused spotlight with a particular concentration,

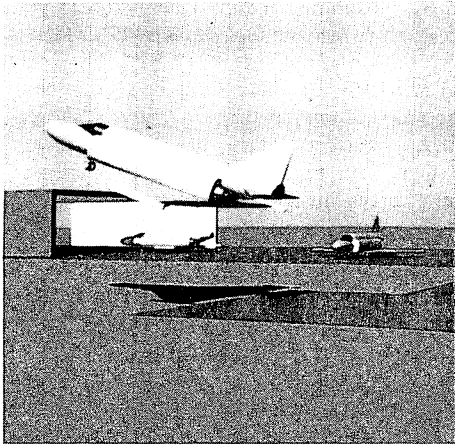


Figure 2

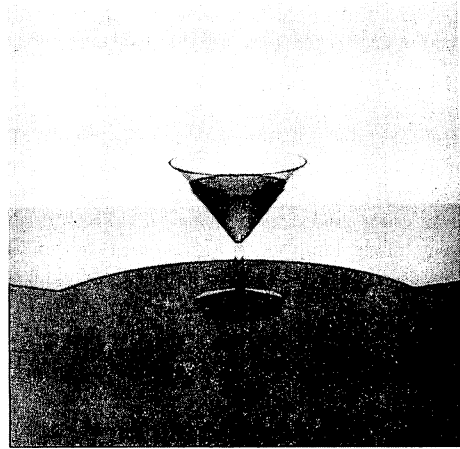


Figure 3

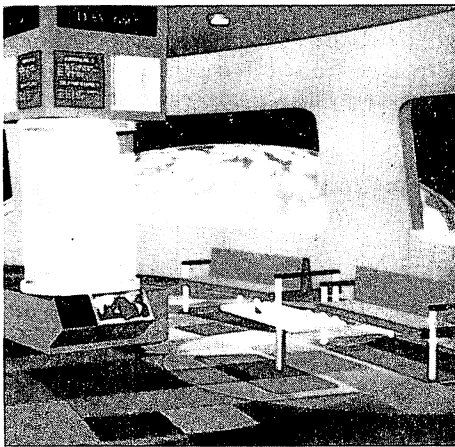


Figure 4

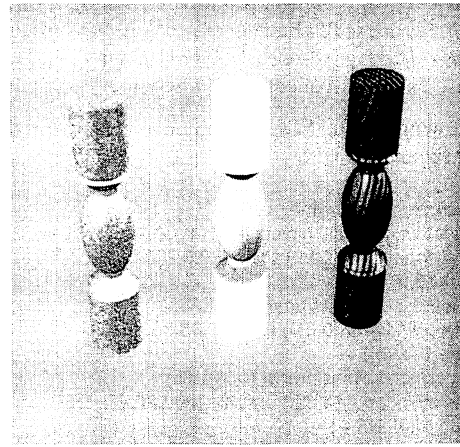


Figure 5

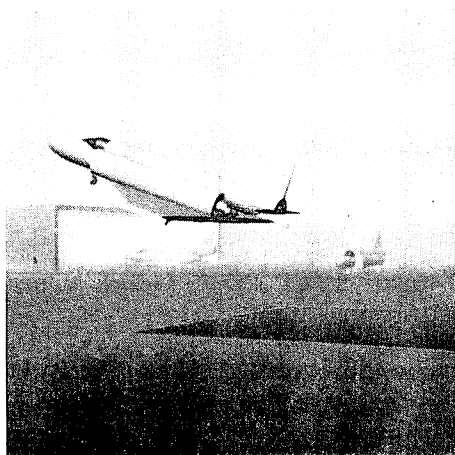


Figure 6

Note: All images were originally in color.

direction, and solid angle [6]. The user may also indicate whether or not each light source should cast shadows.

The FOG statement specifies atmospheric attenuation of light rays. Rays are faded toward the background color as an exponential function of the distance the ray travels. The FOG statement can be used to simulate day or night fog, underwater conditions, haze, or aerial perspective, depending on the background color and fog density. The image shown in Figure 6 was made using the FOG statement with a light gray background color. This image is otherwise identical to Figure 2.

The INCLUDE statement allows scene descriptions to be broken up into several files, with a main description including sub-files as necessary. This makes management of complex scenes easier, particularly in cases like animation, where object descriptions may be the same from scene to scene, with only the positions and orientations changed.

The CLASS statement allows a particular object description to be given a *class name*. Instances of the class may then be used as if they were built-in primitives. Stating it differently, the primitives are really built-in classes. A class instance may be scaled, rotated, positioned, colored, textured, etc. and may also be used as part of another class or object description.

The following is the SDL description used to produce the image in Figure 3:

```
/* Description of the wine glass ... */
Object is (smoothest shiniest trans(1000,1.65)
white cone at (0,100,0) scale (60.5,200,200)
rotate (0,0,-90) +
smoothest shiniest trans(1000,1.65)
white cylinder at (0,0,0) scale (40.5,200,200)
rotate(0,0,90)) +
(smoothest shiniest trans(1000,1.65)
white cylinder at (0,350,0) scale (300,20,20)
rotate (0,0,-90) +
(smoothest shiniest trans(1000,1.65)
white cone at (0,300,0)
scale (400,300,300) rotate (0,0,90) -
smoothest shiniest trans(1000,1.65)
white cone at (0,320,0)
scale (400,300,300) rotate (0,0,90)))

/* ... and of the wine itself ... */
Object is smoothest shiniest trans(1000,1.36) red cone
at (0,320,0) scale (280,210,210) rotate (0,0,90)

/* Description of the backdrop ... */
Object is white block at (0,-20,0) scale (2000,20,2000)
Object is white block at (0,2000,-2000) scale (2000,2000,20)

/* Lighting and camera parameters ... */
Light spot (1,62) intensity 0.8 white
at (0,1000,-100) toward (0,0,-2000)
Ambient intensity 0.30 white
Background light gray
Camera at (0, 1500, 4000) target at (0, 400, 0)
```

3. Scene Compilation

The PRCOMP program uses a LALR(1) parser, generated by the UNIX YACC utility, to parse the SDL description and build internal CSG expression trees for object and class descriptions. All instances of user-defined classes are

expanded by substituting the class definition in place of the instance. The intermediate file output by PRCOMP is entirely in terms of built-in primitives. PRCOMP also expands m-ary union and intersection operators into balanced trees of binary union and intersection operators. Analysis shows that ray tracing these balanced binary trees is considerably more efficient than ray tracing the m-ary operators directly (logarithmic versus linear time complexity).

PRCOMP uses the location, rotation, and scaling information provided in the SDL description to generate a transformation matrix for each primitive (leaf node) in the output CSG description. PORTRAY uses this matrix to transform rays from the scene coordinate system to the local coordinate system of a particular primitive during intersection calculations (see section 4). PRCOMP also generates the inverse of this matrix, which is used to transform normal vectors from the local coordinate system to the scene coordinate system.

At its discretion, PRCOMP may generate a bounding volume for a given primitive instance or CSG expression subtree. PORTRAY uses such bounding volumes in order to make quick comparisons between a ray and a CSG expression so that detailed intersection calculations need not be done for rays which obviously do not pass near the object described by the expression. At present, the bounding volumes are boxes aligned with the axes of the scene coordinate system, but some experimentation with other bounding volumes [7] is planned.

4. Rendering

PORTRAY renders an image of a compiled scene description by tracing a ray from each image pixel to the scene [8]. These primary rays may generate subsidiary rays upon striking a surface which reflects and/or refracts the ray. This process may continue recursively to produce a tree of rays, whose depth is controlled by an adaptive scheme [9] and by a "hard" depth limit. Additional shadow rays are traced from each surface intersection to each shadow-casting light source, in order to determine whether or not light from the source reaches the intersection point on the surface.

Anti-aliasing is performed by adaptively supersampling when adjacent pixel values differ sharply [8]. This anti-aliasing technique may overlook very small details that "fall between the cracks", but is much less expensive than supersampling throughout the image. Recently introduced stochastic sampling techniques [10,11] are being considered as an alternative anti-aliasing scheme for PORTRAY.

Ray tracing a CSG expression [12] involves a postorder traversal of the CSG expression tree. At each internal (operator) node of the tree, lists of intersections from the left and right subtrees are merged according to the semantics of the operator (union, intersection, or subtraction). When the root of a particular expression tree (object description) is reached, the intersection nearest the ray origin is chosen as the intersection between the ray and the object. In some cases, for example, when FAKE (non-refractive) transparency is specified, PORTRAY uses subsequent intersections in the intersection list to render an object. The earth hologram display in Figure 4 is one application of fake transparency.

As mentioned earlier, intersections between a ray and a primitive are performed by transforming the ray from scene space to a local coordinate system in which the primitive has a simple, canonical form. For example, the spherical primitive is always a unit sphere centered at (0,0,0) in its local coordinate system, even though it might be positioned at (1000,2000,3000) and stretched into an ellipsoid by unequal scaling in the SDL description. It is straightforward to convert a ray into the local coordinate system using the transformation matrix supplied by PRCOMP. However, it is more difficult to convert the surface normal vector at the intersection point from the local coordinate system back to the scene coordinate system. The problem arises because angles are not preserved by unequal scaling, so that a vector perpendicular to the surface in the local coordinate system may no longer be perpendicular to the surface after the transformation. PORTRAY avoids this problem by generating three points in the plane tangent to the surface, transforming these points from the local coordinate system to the scene coordinate system, and then reconstructing the tangent plane and the normal vector in the scene coordinate system.

PORTRAY uses a number of techniques to improve the performance of the rendering process. In its simplest form, ray tracing is very much a "brute force" technique, since it exhaustively computes all intersections between every ray and every object in the scene. PRCOMP computes a bounding rectangle in image space for each object, so that PORTRAY knows which pixels may contain a direct image of a given object. PORTRAY then uses the bounding rectangles to efficiently determine which objects can be intersected by a primary ray from a given pixel. Other objects are excluded from consideration during the tracing of that primary ray.

The benefits of the bounding rectangles are limited to primary rays. PORTRAY also uses bounding boxes generated by PRCOMP to quickly exclude objects from consideration in tracing *any* given ray. If a ray does not intersect the bounding box of an object, then the ray cannot intersect the object at all. Checking a ray against a bounding box is only slightly faster than generating the intersections between a ray and a primitive. However, bounding boxes really pay off for objects with complex CSG descriptions. A single bounding box test may exclude from consideration an entire tree or subtree, thus saving dozens or hundreds of primitive intersection calculations.

Testing a ray against a bounding box is fruitful only if the test proves negative and the object within the box is excluded from further consideration. If the bounding box test is positive, the program must go on to intersect the ray with the object, so the box test is a wasted operation. (This is why it is desirable for the bounding volume to fit the object as tightly as possible [7].) PORTRAY exploits a property which we call *bounding volume coherence* to reduce the number of positive bounding box tests. Bounding volume coherence is based on the observation that rays traced from adjacent pixels follow similar paths, even down through subsidiary levels of the ray tree. Thus, there is a high probability that a bounding box test which was positive for the previous ray tree will be positive for the current ray tree. When a bounding box test is positive, PORTRAY flags it with a value indicating the ray tree position of the ray being traced. On the next ray, flagged

bounding boxes are *assumed* to test positive at the same position in the ray tree, and the bounding box test is not performed. There is a performance penalty if the assumption is false, but the appearance of the image is unaffected.

PORTRAY ray traces about 10% faster when bounding volume coherence is used. This is particularly interesting, since an attempt to make more general use of ray coherence, reported in [13], indicated that no performance benefit was obtained.

PORTRAY generated the image in Figure 3 at 512×512 resolution in 70 minutes on a Pyramid 90x, tracing a total of 606 thousand rays. The more complex image in Figure 4 was rendered at the same resolution in 324 minutes. Fewer rays (502 thousand) were traced in this case, because fewer pixels contained reflective or refractive objects.

5. Image Format and Tools

At a conceptual level, PORTRAY images are rectangular arrays of pixels. Each image consists of several UNIX files, including an *image description file* (IDF). The IDF is a file of ASCII text which describes the image and each of the other files which form part of the image. Table 1 lists the various files which may exist as part of an image. A particular image need not contain all of these files. The height, width, and depth of each of the image data files is described in the IDF; the data files themselves contain only the pixel intensity information. Data files may optionally be run-length encoded to reduce storage cost.

The multiple-file image representation was chosen to provide a high degree of flexibility in the manipulation of image data. For example, an RGB image with 24 bits per pixel (bpp) would be stored in three separate files, each with 8 bpp. The red, green, and blue data could be accessed separately or as a single RGB image. The format of the IDF reinforces this flexibility, since the IDF can be modified with an ordinary text editor when special handling is needed. Thus, it is not always necessary to build a new image manipulation tool, even when unforeseen needs arise.

Table 1: Image File Types	
Filename Suffix	Description of Contents
idf	image description file
red	red image data
grn	green image data
blu	blue image data
cvg	pixel coverage data [15]
gry	gray-scale image data
lut	color lookup table (LUT)
enc	image encoded for LUT
log	history of image (text)

Several tools have been developed to process PORTRAY images, including the following:

- *iencode*, which generates an 8 bpp image using a given color lookup table, from a 24 bpp RGB image, using the algorithm described by Heckbert [14].

- *ilut*, which generates a color lookup table containing colors which are appropriate for displaying a given 24 bpp RGB image. The lookup table is generated from a color histogram of the RGB image, using the "median cut" color space subdivision algorithm, also described in [14]. *ilut* and *iencode* are used to prepare a 24 bpp image for display on an 8 bpp color graphics system, such as an AED terminal or a Sun workstation. PORTRAY generates all images in 24 bpp RGB form.
- *icomp*, which combines images according to a specified composition operator [15]. Figure 4 is an example of a composite image produced using *icomp*. The starfield visible through the space station windows was separately generated, and then composited with the PORTRAY image of the space station and planet.
- traditional image processing algorithms, including histogram equalization and gamma correction, are used to process texture images and to adjust contrast, brightness, and color of images prior to photographing them with a film recorder.

6. Conclusions

PORTRAY is a flexible image synthesis system which derives much of its power from a high-level scene description language. Constructive solid geometry is an excellent, easy-to-use geometric modelling technique for man-made objects, but is less appropriate for natural objects. Ray tracing is an expensive rendering technique, but is well suited to CSG models and is capable of simulating a wide variety of optical phenomena. PORTRAY incorporates various techniques for speeding up the ray tracing of CSG models, including a novel technique for exploiting bounding volume coherence. To further speed up ray tracing, we are planning to experiment with parallel ray tracing algorithms on an experimental 16-processor INMOS Transputer system, being constructed at the University of Saskatchewan.

Acknowledgements

This research could not have been performed without the support and facilities of the Department of Computational Science and the University of Saskatchewan.

References

- [1] Peachey, D.R., "PORTRAY - An Image Synthesis System for Realistic Computer Graphics", Research Report 84-18, Dept. of Computational Science, University of Saskatchewan, December, 1984.

- [2] Tilove, R.B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems", *IEEE Trans. Computers* C-29, 10 (Oct. 1980), 874-883.
- [3] Berk, T., Brownston, L. and Kaufman, A., "A New Color-Naming System for Graphics Languages", *IEEE Computer Graphics & Applications* 2, 3 (May 1982), 37-44.
- [4] Cook, R.L. and Torrance, K.E., "A Reflection Model for Computer Graphics", *ACM Trans. Graphics* 1, 1 (Jan. 1982), 7-24.
- [5] Peachey, D.R., "Solid Texturing of Complex Surfaces", *Computer Graphics* 19, 3 (July 1985), Proceedings of SIGGRAPH '85, 279-286.
- [6] Warn, D.R., "Lighting Controls for Synthetic Images", *Computer Graphics* 17, 3 (July 1983), Proceedings of SIGGRAPH '83, 13-21.
- [7] Weghorst, H., Hooper, G., and Greenberg, D.P., "Improved Computational Methods for Ray Tracing", *ACM Trans. Graphics* 3, 1 (Jan. 1984), 52-69.
- [8] Whitted, T., "An Improved Illumination Model for Shaded Display", *Comm. ACM* 23,6, 343-349.
- [9] Hall, R.A. and Greenberg, D.P., "A Testbed for Realistic Image Synthesis", *IEEE Computer Graphics & Applications* 3, 8 (Nov. 1983), 10-20.
- [10] Cook, R.L., Porter, T., and Carpenter, L., "Distributed Ray Tracing", *Computer Graphics* 18, 3 (July 1984), Proceedings of SIGGRAPH '84, 137-145.
- [11] Dippe, M.A.Z. and Wold, E.H., "Antialiasing Through Stochastic Sampling", *Computer Graphics* 19, 3 (July 1985), Proceedings of SIGGRAPH '85, 69-78.
- [12] Roth, S.D., "Ray Casting for Modeling Solids", *Computer Graphics & Image Processing* 18 (1982), 109-144.
- [13] Speer, L.R., DeRose, T.D., and Barsky, B.A., "A Theoretical and Empirical Analysis of Coherent Ray-Tracing", *Proceedings of Graphics Interface '85*, 1-8.
- [14] Heckbert, P.S., "Color Image Quantization for Frame Buffer Display", *Computer Graphics* 16, 3 (July 1982), Proceedings of SIGGRAPH '82, 297-307.
- [15] Porter, T. and Duff, T., "Compositing Digital Images", *Computer Graphics* 18, 3 (July 1984), Proceedings of SIGGRAPH '84, 253-259.

**An Adaptive Subdivision by Sliding Boundary Surfaces
for Fast Ray Tracing**

Keiji NEMOTO and Takao OMACHI

C&C Systems Research Laboratories, NEC Corporation

4-1-1 Miyazaki, Miyamae-ku, Kawasaki, Kanagawa, 213 Japan

(044) 855-1111

ABSTRACT

This paper presents an adaptive subdivision algorithm for fast ray tracing implemented on parallel architecture using a three dimensional computer array. The object space is divided into several subregions and boundary surfaces for the subregions are adaptively slid to redistribute loads of the computers uniformly. Since the shape of the subregions is preserved as orthogonal parallelepiped the redistribution overhead can be kept small. The algorithm is quite simple but can avoid load concentration to a particular computer.

Simulation results reveal that the adaptive space subdivision algorithm by sliding boundary surfaces reduces the computing time to $3/4-1/5$ as much as that for the conventional space subdivision algorithm with no redistribution, which reduces the computing time almost proportionally to the number of the computers.

KEYWORDS: sliding, adaptive, parallel, ray tracing, subdivision, boundary.

Introduction

Among general image synthesis methods available today, ray tracing¹ is probably the most realistic technique, because it models a wide range of natural phenomena. However, it requires a large amount of computing time. The calculation for ray-object intersections requires 75-95 percent of the total computing time¹.

Various approaches have been attempted toward speeding up of ray tracing. Previous research reports are categorized as follows:

(1) Multicomputer system by image subdivision²: An image to be generated is divided into several subimages and each of the computers generates one or more subimages independently.

(2) Vectorization³: Since the ray-object intersection calculations belonging to the different pixels and the intensities of the

different pixels are calculated completely independently, the calculations can be vectorized.

(3) Space subdivision^{4,5,6}: The three dimensional space of a scene to be rendered is divided into subregions. The rays which are cast into each subregion are tested for intersection with the objects contained within the subregion. Data on rays that exit a subregion are passed to the appropriate neighbor.

The first two ways do not reduce the number of ray-object intersection calculations, but speed up the intersection process itself, by parallel processing and specialized hardware.

On the other hand, the space subdivision method can reduce the number of calculations, because it tests rays for intersection only with the objects contained within the subregions that rays pass through, instead of all objects in the entire scene.

Recent work has applied a parallel architecture to this space subdivision algorithm⁴. This architecture uses a three dimensional computer array, each computer of which is assigned to one or more subregions. The shapes of the subregions are "general cubes", which are general hexahedron, and the shapes are adaptively controlled to realize a roughly uniform load distribution. This algorithm has the following problems:

1) Load is transferred among the subregions by moving corners of a general cube indicating the subregion. The moving operation to distribute the load is quite difficult because moving one corner affects the loads of the eight subregions holding it in common. The problem is how to select the corner to be moved and how to determine the direction and the length to move the corner in a three dimensional space in order to distribute the loads of the eight subregions at once.

2) When rays exit the subregion, the neighboring subregion that rays are passed to is determined by boundary-intersection calculation. However, boundary-intersection testing for general cubes is a significant overhead.

3) When a corner of the subregion is moved, an

appropriate part of the object descriptions contained within the subregion are determined and pertinent data is passed to the neighboring subregions. This operation is as expensive as boundary-intersection testing.

A moving corner method would greatly affect the performance of this algorithm. However, the problem how to move the corner to distribute the load cannot be easily solved because of the difficulties mentioned above.

This paper presents a new approach to solve the problems above. The shapes of the subregions are limited to orthogonal parallelepipeds, and load is transferred by sliding boundary surfaces of the subregions.

The following sections will discuss a simple subdivision algorithm for parallel architecture and give simulation results.

New Space Subdivision Algorithm

The essential algorithm characteristics are:

- 1) The three dimensional space of a scene to be rendered is divided into several subregions by planes perpendicular to a coordinate axis(Fig.1). Positions x_i , y_i , and z_i of the dividing planes have integer coordinate values. Thus, each subregion is an orthogonal parallelepiped which consists of several unit cubes. A unit cube is a cube whose size is 1 and whose edges are parallel to each coordinate axis.
- 2) Each computer of three dimensional computer array is assigned to one subregion and maintains only the object descriptions contained within that subregion.
- 3) Each computer has 6 connections to neighboring computers in order to pass messages

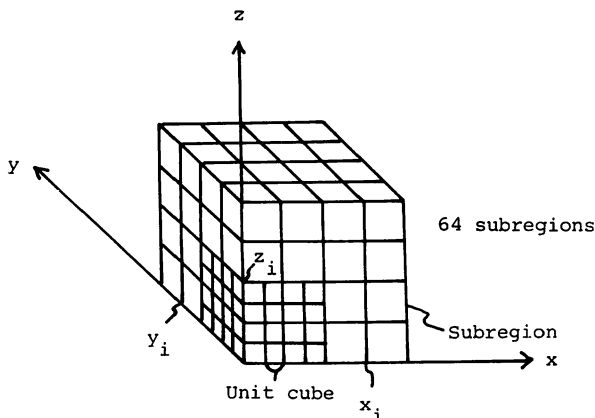


Figure 1. Three dimensional space division

(Fig.2), which consist of information about rays and redistribution. Each computer also has a direct connection to a host computer and to a frame buffer. Messages regarding object descriptions are directly sent from a host computer to all computers as broadcast messages. Each computer determines which object is contained within its own subregion and preserves only its description.

4) Initial rays from the eye point are created by all computers in parallel. An image to be generated is divided into subimages and each computer is assigned to one of the subimages. Each computer creates the initial rays which pass through its own subimage. Then each initial ray is transferred to the appropriate subregion where the initial ray starts. After each of the rays has reached to the appropriate subregion, it is tested for intersection with those objects within the subregion.

5) Rays that exit the subregion are passed to neighboring subregions via connection between computers. The three dimensional digital line⁶ is generated for efficient tracing of rays. The three dimensional digital line is an array of unit cubes pierced by the ray (Fig.3).

To determine the the array of unit cubes, two DDAs (Digital Differential Analyzers)⁶ generate two digital lines synchronously which are the projections of the three dimensional digital line to two coordinate planes. The error values of two DDAs are compared to decide the correct direction of the three dimensional digital line (Fig.4).

Since the subregion consists of several unit cubes and the three dimensional digital line is generated only by addition of integer values, rays can rapidly traverse the subregion and also enter the appropriate neighboring subregion by using the three dimensional digital line.

6) For redistribution, the boundary surface between two subregions is slid by one unit and a

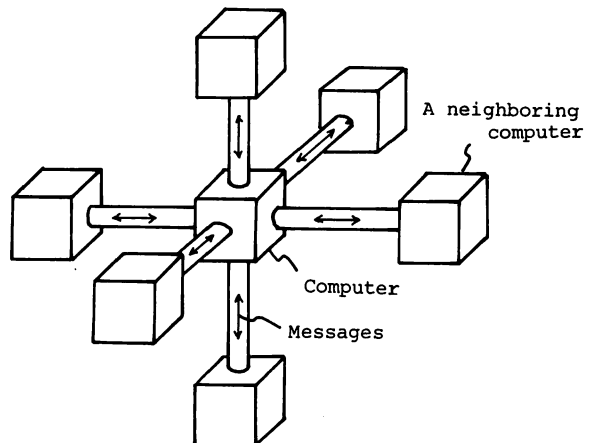


Figure 2. 6 connections from a computer

part of the load for one subregion is transferred to the other subregion. The following section gives more details about sliding boundary surfaces.

Sliding Boundary Surfaces

The adaptive subdivision by sliding boundary surfaces is as follows:

1) At the beginning, one of three coordinates axes is set as a driving axis (e.g. x axis in Fig.5). Boundary surfaces for the subregion perpendicular to the driving axis are slid by one unit along the driving axis to transfer the load from one subregion to a neighboring subregion.

The subregion load is related to the number of the objects contained within the subregion.

Therefore, the axis along which the numbers of the objects in the subregions are most varied is set as a driving axis.

2) Each computer counts the running time while the computer actually processes the rays. Each computer also counts the waiting time while the computer has no ray to be processed and is waiting the rays passed from the neighboring computers. The ratio (running time) / (waiting time) is defined as a parameter to indicate the load for the subregion.

3) For redistribution, loads for two subregions on both sides of the boundary surface are compared by the computers assigned to these two subregions at intervals of the given time . If the load for one subregion is lower than that for the other subregion and the lower load is under the given threshold value, the boundary surface is slid

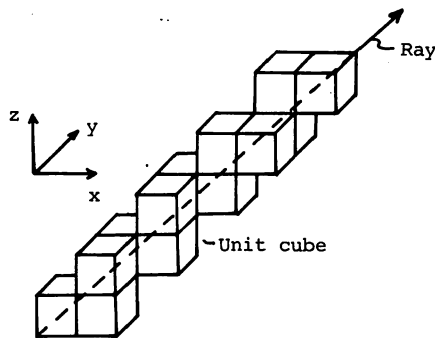


Figure 3. Three dimensional digital line

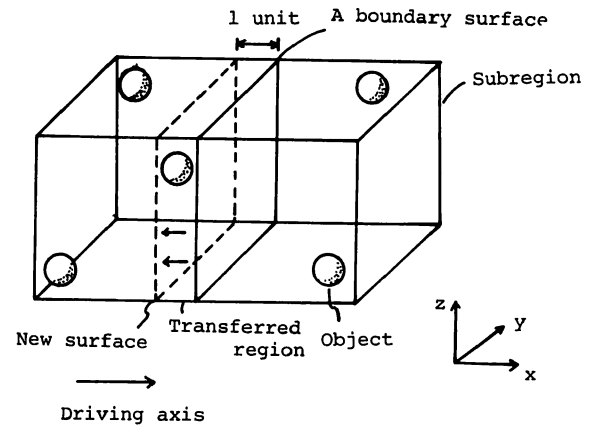


Figure 5. Sliding a boundary surface

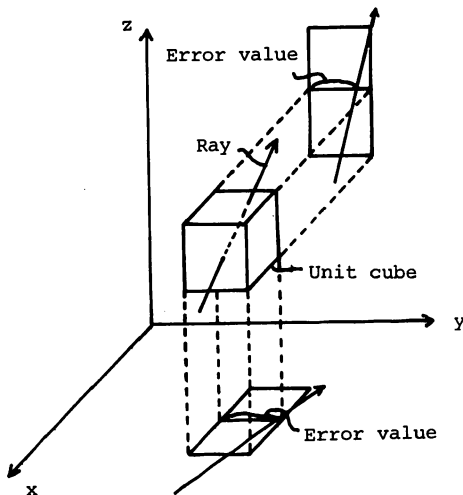


Figure 4. Error values of two DDAs

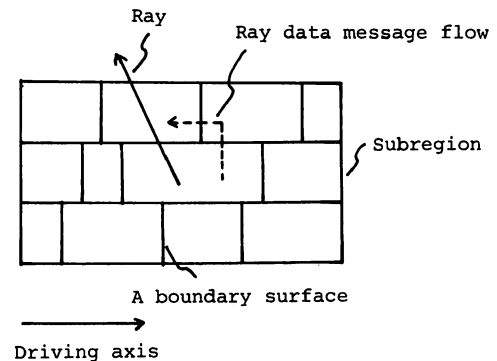


Figure 6. Boundary surfaces discrepancy
(Two dimensional view)

by one unit along the driving axis from the lower load subregion to the higher load subregion. By this operation, a part of the subregion with the higher load (called the transferred region) is cut away and added to the lower one. Simultaneously, the object descriptions contained within the transferred region are transferred. Thus, the load is simply and efficiently transferred. These operations are locally executed by the computers assigned to these two subregions.

4) As each boundary surface is slid for the redistribution, some discrepancy in the boundary surface occurs (Fig.6). However, since the connections between computers are fixed, there could be a case wherein a computer has no direct connection to another computer assigned to the neighboring subregion. In this case, data concerning the rays that exit the subregion cannot be directly passed to the appropriate computer assigned to the appropriate subregion, so that data on rays are passed to the direct connected computer which is assigned to the subregion located on the same driving axis with the appropriate subregion (shown by a dotted line in Fig.6). After that, data on rays are passed along the driving axis where they can finally reach the appropriate computer.

Essential characteristics of the method are as follows:

1) For redistribution, only the loads for two subregions on both sides of the boundary surface perpendicular to the driving axis are compared. Thus, the redistribution between the subregions is easily determined.

2) The shape of the subregions is preserved as an orthogonal parallelepiped. Since boundary surfaces are rectangular and perpendicular to coordinate axes, boundary-intersection testing is a small overhead.

3) Only the boundary surfaces perpendicular to the fixed driving axis are slid along the axis by one unit and the object descriptions contained within that transferred region are transferred. So the redistribution does not cause a significant overhead.

4) Since the given threshold value stops the sliding between the highly loaded subregions, a thrashing whereby the object descriptions are repeatedly moved between the highly loaded computers is avoided. A thrashing between the lightly loaded subregions does not matter to the total performance.

5) Because of the simplicity of this method, it can be easily implemented on a three dimensional computer array.

Results

The proposed adaptive subdivision algorithm by sliding boundary surfaces is simulated to evaluate the redistribution effect. The simulation results for redistribution are compared with that for no redistribution when the load is concentrated to some particular computers.

Beforehand, the effect of the conventional space subdivision algorithm without redistribution is evaluated by simulation.

A. Simulation Methods

The algorithms have been written in a C program and tested on a Vax-11/780 under the Unix operating system. A parallel process simulator⁷ has been created to evaluate the algorithms. The simulator virtually causes the computers to run in parallel and counts the running time and the waiting time of each computer to calculate the load. The simulator also counts the computing time for generating an image by parallel architecture.

Objects are only spheres. These spheres are described by their center position, radius, color, and reflecting parameters and these parameters are generated as uniform random numbers. Therefore, the objects are located in a space at random.

B. Space Subdivision Effect

First, the effect of the conventional space subdivision algorithm is evaluated.

Figure 7 shows the computing time of the space subdivision algorithm without redistribution.

Result A shows the computing time when only a single computer is assigned to all subregions. The algorithm implemented on a single computer reduces the computing time on the order of $S^{2/3}$ (S = the number of the subregions).

Result B shows the computing time when each computer of three dimensional computer array is assigned to one subregion but initial rays are created by the computer whose subregion contains the eye point. The difference between results A and B means the parallel processing effect using the three dimensional computer array.

Result C shows the computing time when the initial rays are also created by all computers in parallel. This space subdivision method with the parallel initial ray creation reduces the computing time on the order of $S^{1.5}$. The difference between results B and C means the parallel creation effect of the initial rays. The parallel initial ray creation is effective when S is large.

Figure 8 shows the computing time when the number of objects is changed in the case of result C in Fig. 7. The computing time can be reduced on almost the same order even if the number of the objects is changed.

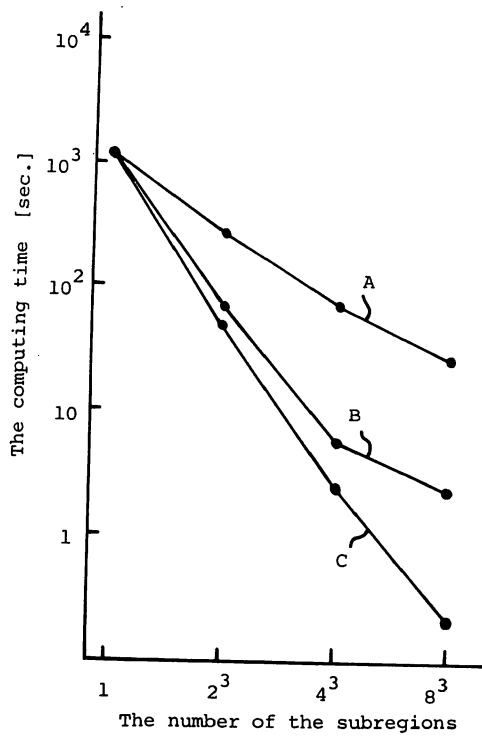


Figure 7. Space subdivision effect

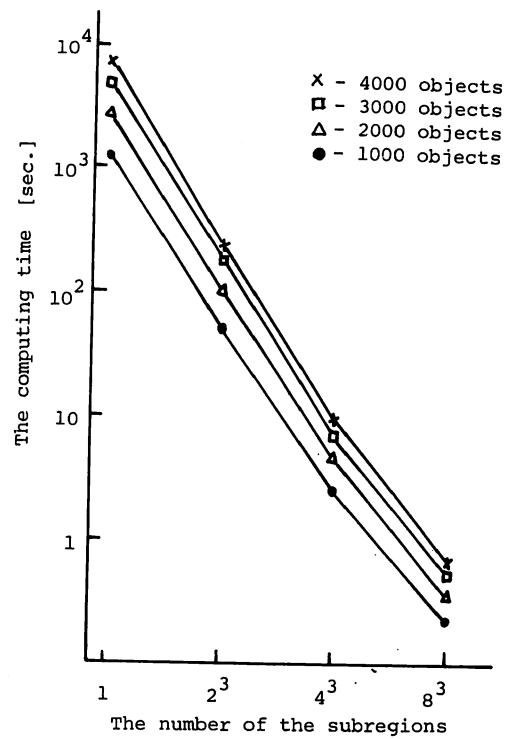


Figure 8. Space subdivision effect for several numbers of the objects

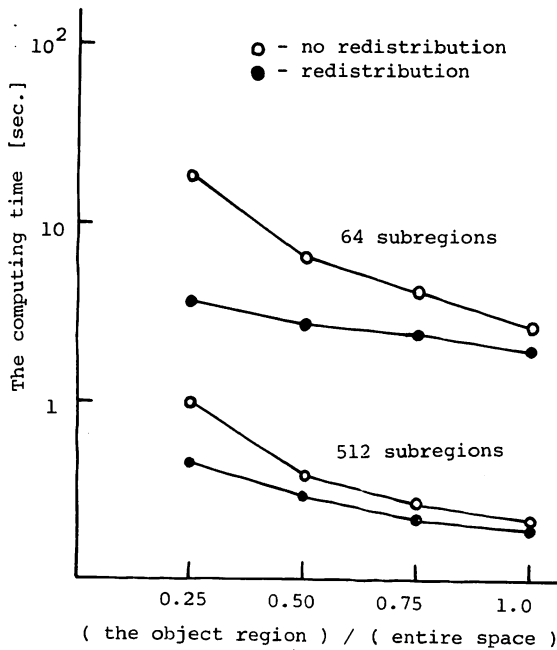


Figure 9. Redistribution effect for 1000 objects

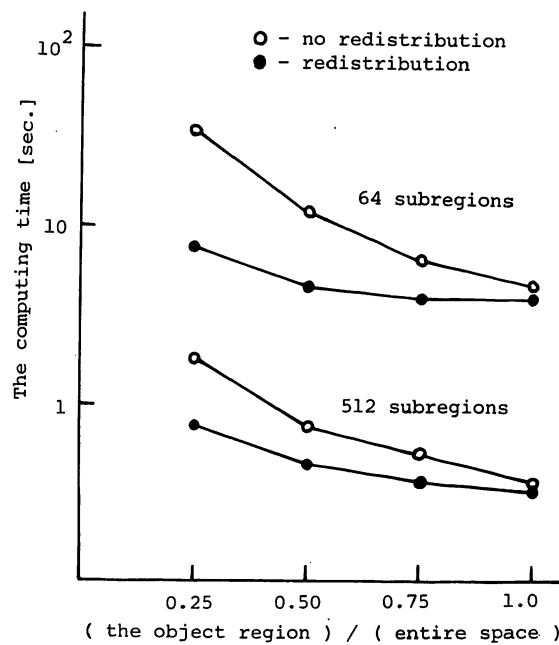


Figure 10. Redistribution effect for 2000 objects

C. Proposed Redistribution Effect

In order to examine the redistribution effect, the region where all objects are located is reduced from the entire space to quarter of the entire space. The smaller region the objects are located in, the more load is concentrated to some particular subregions.

Figures 9 and 10 show the comparisons between the computing time for redistribution by sliding boundary surfaces and that for no redistribution. The horizontal axes of Figs. 9 and 10 means the volume ratio of the region where all objects are located against the entire space.

When the objects are uniformly located in a space at random, the loads have been almost uniformly distributed initially so that the redistribution does not work so effectively. Even so, the redistribution can reduce the computing time to $3/4$ as much as that for no redistribution.

The more concentrated the objects and the loads are to a part of the subregions, the greater the redistribution effect becomes. The effect becomes up to $1/5$ when the objects are concentrated to the quarter of the entire space. these results mean that the redistribution by sliding boundary surfaces has an equivalent effect to distribute the concentrated objects to the entire space.

Moreover, Figs. 9 and 10 show almost same redistribution effect, so that not the number of the objects but the objects location in the space controls the redistribution effect.

Conclusions

This paper has presented a simple adaptive subdivision algorithm implemented on the parallel architecture using a three dimensional computer array. Boundary surfaces of the subregions are adaptively slid to redistribute loads of the computers uniformly. Since the shape of the subregions is preserved as orthogonal parallelepiped the redistribution overhead can be kept small. By using this algorithm, the computing time is reduced as much as $3/4$ - $1/5$ of that for no redistribution.

Acknowledgment

The authors would like to express appreciation for continuous encouragement from K.Niwa. They also acknowledge the significant contributions of S.Fukui and H.Kanazawa.

References

1. T.Whitted, "An Improved Illumination Models for Shaded Display," Comm. ACM, Vol.23, No.6, '80, pp.343-349.
2. H.Nishimura, H.Ohno, T.Kawata, I.Shirakawa, and K.Omura, "LINKS-1: A Parallel Pipelined Multimicrocomputer System for Image Creation," Proc. of the 10th Symp. on Computer Architecture, SIGARCH, '83, pp.387-394.
3. D.Plunkett and M.Bailey, "The Vectorization of a Ray-Tracing Algorithm for Improved Execution Speed," IEEE CG&A, Aug. '85, pp.52-60.
4. M.Dippe and J.Swensen, "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," Computer Graphics, Vol.18, No.3, Jul. '84, pp.149-158.
5. A.Glassner, "Space Subdivision for Fast Ray Tracing," IEEE CG&A, Oct. '84, pp.15-22.
6. A.Fujimoto and K.Iwata, "Accelerated Ray Tracing," Proc. of CG Tokyo '85, T1-2.
7. C.Binding, "Cheap Concurrency in C," SIGPLAN Notices, Vol.20, No.9, Sep. '85.

Profiling Graphic Display Systems

Peter Schoeler

Alias Research Incorporated
Toronto, Ontario

and

Department of Computer Science
University of Toronto

Alain Fournier

Computer Systems Research Institute
Department of Computer Science
University of Toronto
Toronto, Ontario
M5S 1A4

ABSTRACT

Graphics systems using three dimensional models, and computing a colour shaded image for a raster display are very common, and range widely in performance and cost. Despite the numerous variations in rendering techniques, visibility determinations, illumination models and modeling primitives manipulated, it is important to be able to compare them when rendering similar scenes.

We present here the first results of a series of profiling of different rendering systems displaying the same scenes on the same machine. The systems studied are a ray-caster, a system using a depth-buffer for visibility determination, and a system using a scan-line Watkins algorithm. The first and last systems have an antialiasing option. Two types of scenes were used, one made of a constant number of polygons varying in size, and the other made of parametric surfaces varying in level of subdivision.

The results, mainly useful for relative comparisons, confirm some predicted behaviours. The depth-buffer algorithm degrades considerably when the depth complexity increases. The ray-caster is not much influenced by the number of polygons, but by the total number of pixels covered. The most striking result is the large proportion of time spent on shading. It is a strong indication that work on ways to make shading computations less expensive, and to design special hardware for that purpose would be fruitful.

KEYWORDS: display systems, rendering techniques, profiling, shading, visibility determinations.

RESUME

Les systèmes graphiques qui utilisent des modèles à trois dimensions et qui produisent des images ombrées en couleur pour des affichages *rasters* sont maintenant très répandus et diffèrent énormément en puissance et en coût. En dépit des grandes variations dans les techniques de détermination de visibilité, les techniques d'ombrage et les techniques de modelage qu'ils utilisent, il est important de pouvoir comparer leurs performances quand ils rendent la même scène.

Nous présentons ici les premiers résultats d'une série de profilage de différents systèmes d'affichage produisant les mêmes scènes sur la même machine. Les systèmes étudiés sont un *lanceur de rayon*, un système utilisant une *mémoire de profondeur* pour déterminer la visibilité, et un système utilisant l'algorithme de Watkins avec la conversion en ligne de balayage. Le premier et le dernier système ont tous les deux une option d'*antialiasing*. Deux genres de scènes ont été utilisées. Un était fait d'un nombre constant de polygones dont seule la taille changeait, et l'autre de surfaces paramétriques à des niveaux variés de subdivision.

Les résultats, surtout utiles pour des comparaisons relatives, confirment beaucoup de prévisions. La performance de l'algorithme de mémoire de profondeur se dégrade de façon considérable quand augmente la complexité de profondeur. Le lanceur de rayon n'est pas très influencé par le nombre de polygones, mais plutôt par le nombre total de pixels recouvertes. Le résultat le plus frappant est la grande proportion de temps consacrée aux calculs d'ombrage. C'est une forte indication du fait que plus de recherches pour améliorer l'efficacité de ces calculs et pour développer du matériel pour cet effet pourrait s'avérer payant.

MOTS CLÉS: systèmes d'affichage, techniques de rendu, profilage, ombrage, détermination de visibilité.

1. Motivations

A graphic display system, in the context of this study, is a combination of hardware and software which extracts object descriptions from an application database, applies geometric transformations to create instances of objects, determines their projections in a two-dimensional screen space, and computes the colour value of each pixel for the frame buffer of a raster display device. We will limit ourselves to the consideration of systems which handle three-dimensional models of objects, and aim at a *realistic* picture. Even with these restrictions, there exist systems which vary in performance from real-time to real-long-time (several hundred hours per frame), and from a few thousand dollars to a few million.

A display system has three main components (note that we are not considering the interaction with the user in this

study). The first one is the modelling component, which is really part of the application. By modelling here we do not mean the designing and creation of the models, but their retrieval and/or generation on the fly. For example extracting polygons from the database, computing points on a parametric surface, generating stochastic data are all modelling operations. The second component involves geometric operations. This includes clipping, perspective transformations, mapping to the screen. Two other important parts of the geometric operations are visibility determinations and shading. They are classified within the geometric operations because they use directly the geometric properties of the objects and the scenes for their computations and none of the screen geometric properties. And finally the third component includes all the display operations. In a raster system they are mainly the "scan-conversions", the sampling and filtering operations, and writing out the image (to a file or directly to the frame buffer).

It is important to note that this subdivision is independent of the rendering scheme. For instance, consider a depth-buffer system and a ray-caster. They could have the same modelling primitives and operations, such as B-spline surfaces and adaptive subdivision; the geometric operations for the depth-buffer system are mainly as described above, and consist of ray intersection calculations and shading for the ray-caster; the display operations are scan conversions and depth comparisons for the depth-buffer, and distributing into "scan buckets", subdividing the screen, etc., for the ray-caster.

An indirect confirmation of the validity of this view is that when new algorithms or new hardware appear, they can easily be categorized following this scheme.

New modelling techniques are appearing regularly [FoFC82, Reev83, Gard84, Gard85]. In geometry, the basic operations do not change very much, but the shading techniques became more sophisticated and more expensive [Cook81]. The visibility problem remains a active area of research, and even more effort is expended to make it more difficult [Whit80]. In this respect ray-tracing and ray-casting are properly rendering methods, that is they involve the whole rendering scheme. Therefore they include the modelling, geometric and display operations. Recently the display side (notably the sampling and filtering operations) have received the most attention within that technique [Aman84, CoPC84, DiWo85, LeRU85].

Hardware development, besides the wholesale implementation in hardware of the graphics display system for real-time flight simulators [Scha83], has seen attacks on specific components: the purely geometric operations [Clar82] or the scan conversion component [FGHS85]. The design of specialized hardware for modelling, especially complex modelling, has been only started [PiFo84]. Notable by its absence is the lack of hardware design for shading.

For most systems the goal is the greatest amount of realism for the least cost (in time and hardware to run it on). Given this, it is surprising that the literature is not more abundant on the performance evaluation of such systems.

The information available so far, besides various raw timings for pictures ("Figure X took 450 hours of Vax time") is limited to profiling results on one particular system [ReB185] and numbers and analysis of the performance of visibility determination algorithms [SuSS74]. Crow compared the times spent on modelling, geometric operations, shading and filtering [Crow81], which was mainly oriented towards a comparison of the latter operations.

While most of the work in performance analysis bore on visibility determination, there was mounting evidence that the cost of modelling, and even more shading was rapidly getting larger. Already Crow pointed out that trend in [Crow81]. The result of that is that we have to consider carefully the illumination models and the shading methods, especially as they relate to the visibility algorithms and the display operations. A fast visibility algorithm will degrade in performance if the depth complexity increases and it continues computing the shade for many invisible areas. At this point, an algorithm that computes the shading only for the visible surfaces might win, even if the visibility determination is less efficient.

The first task in comparing various systems is choosing the scene they will be run on. Here again it is a fairly complex problem, with not as many published results as its importance and interest require. Kaplan and Greenberg [KaGr79] and Parke [Park80] addressed the problem for the analysis of depth buffer algorithms in conjunction with various processor architectures. Schmitt [Schm81] did the same, but this time to determine empirically the complexity of various visibility algorithms. More recently Whelan [Whel85] considered the problem again within the context of multiprocessor architectures.

Of course the problem of choosing the right test data is not unique to graphics. The problem here is twofold. One problem is to determine how to measure scene characteristics, and the other is to decide what are the characteristics of "typical" scenes.

2. Methodology

For this first report we tried to keep the number of variables under control, but to have enough variety and relevance to be of use to practitioners. The tactic we have adopted is to have three different renderers displaying the same scenes on the same hardware. The difference between the renderers is mainly in their methods to determine visibility.

The first renderer is a ray-caster, which we will call RC¹. To speed up ray intersection, it subdivides screen space into buckets, and each polygon is added to a bucket list if its bounding box intersects the bucket. For each ray only the polygons listed with the bucket intersected by the ray are examined. It has an antialiasing mode, where pixels are adaptively subdivided if the shades at each corner differ by more than a given threshold. It can adaptively

1. The ray-caster is based on software originally written by Mike Sweeney at the University of Waterloo Computer Graphics Laboratory. An improved version is now a component of the Alias 1 rendering module.

subdivide parametric surfaces that way, but this was not used here to allow easier comparisons.

The other two renderers share the same front end, known as *3d* [AmBr86] at the University of Toronto Dynamic Graphics Project. The second renderer, which we will designate as DB, uses a file depth-buffer to determine visibility. It does not have an anti-aliasing option.

The third renderer uses Watkins algorithm [Watk70] to compute a scanline per scanline visibility. We will call it WS. It has an antialiasing option which uses the full precision in the X direction, and four subscanlines in the Y direction.

They both use a variety of rendering options, with a choice of illumination model, and include texture mapping, except for DB.

2.1. Modelling Primitives

Since the systems have to render the same scenes, they will have to use the same modelling primitives. They are polygons and B-splines patches, the most prevalent in current practice. The scene description actually is defined in a *scene description language*, and various filters generate the files for each renderers.

2.2. Geometric Primitives

Even though it was not mandatory for this study, the three systems all use triangles internally as geometric primitives.

	P1	P2	P3
Modelling Polygons	244	244	244
Geometric Triangles	488	488	488
Effective Triangles	485	485	482
Average Depth	0.60	2.32	8.32
Average Pixels Covered	321	1253	4526
Average Area	326	1313	5376

Table 1. Scene characteristics for polygons

2.3. Display Primitives

The three renderers produce a raster image, and were all set to output the image to a run-length encoded file. They therefore have basically the same output method. They were set to output a 512x512 image of 8 bit each of red, green and blue pixels.

3. Scene Characteristics

In order to isolate only a few variable, we decided to keep the number of modelling primitives constant for each series of scenes. In the first series, we distributed 40 cubes (6 faces each) roughly uniformly over the window. The spacing was chosen so that there was little overlap

between cubes. Then for the subsequent scenes the cubes were linearly doubled around their centres so that the depth complexity, the average area of the polygons and the number of covered pixels all increased regularly.

In the second series, we designed a "glass" made of a 6 by 6 array of B-spline patches, and made three copies of it. There are therefore 3 primitives if primitives are control point networks, but 108 primitives if each patch is considered a primitive. The level of subdivision was set at 2, 4, 8 and 16 segments to a side. In this series the depth complexity and the total number of pixel covered is practically constant. The number of geometric primitives increases and the average size of each decreases to keep the product almost constant. Table 1 and 2 gives the main numbers for each series. Figures 1 to 3 and 4 to 6 are line drawings of the first six scenes.

The statistics given here were chosen to indicate the complexity of the scene. The depth complexity is the average number of object in one pixel, and will allow to gauge the efficiency of visibility determination and of shading. The average area of the polygons, computed analytically from the screen coordinates of the vertices, will help in determining the "polygon set-up time" vs the cost of pixel calculations. A pixel is deemed covered by a polygon if its center is inside the polygon. For the scenes used here the last two numbers are almost equal, but as the polygons become thinner, the difference can become important. Other statistics which are not included here can also be important. The number of edges, and the number of pixels containing edges is an example. In further studies about the role of filtering and antialiasing, we will have to consider them, as well as distinguish between silhouette edges and internal edges. If the scenes are used to test parallel algorithms, the distribution of the primitives in space, and their aspect ratio should be taken into account.

	V1	V2	V3	V4
Modelling Patches	108	108	108	108
Geometric Triangles	864	3744	15552	63360
Effective Triangles	864	3744	15552	63358
Average Depth	0.61	0.59	0.59	0.59
Average Pixels Covered	185	41	9	2
Average Area	185	41	10	2.4

Table 2. Scene characteristics for B-splines

The scenes were all lit by three local light sources. The illumination model used was the same across systems, being the Lambert cosine law for the polygons, and Phong illumination model for the patches, with a *shininess* of 50. The backfacing polygons were not culled, and every polygon was uniformly shaded (no Gouraud shading).

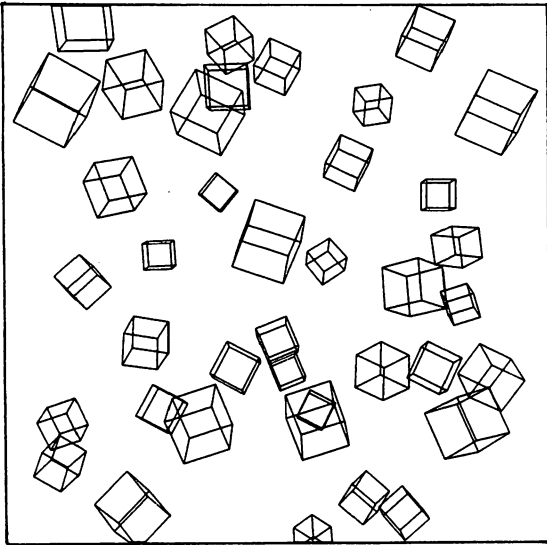


Figure 1. Scene P1

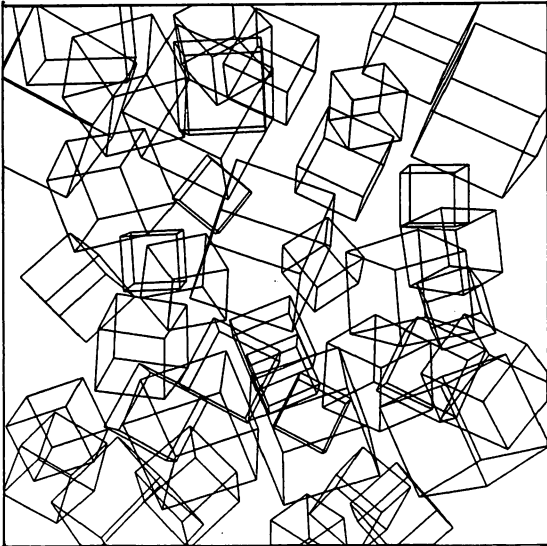


Figure 2. Scene P2

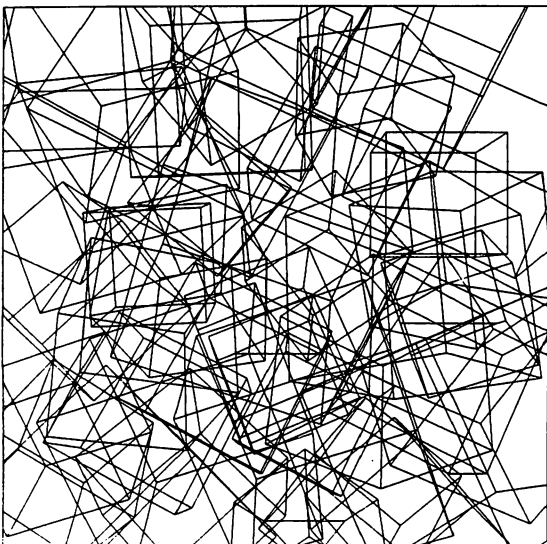


Figure 3. Scene P3

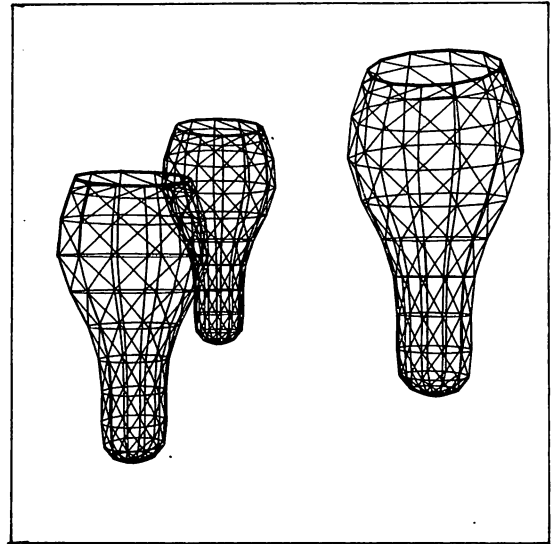


Figure 4. Scene V1

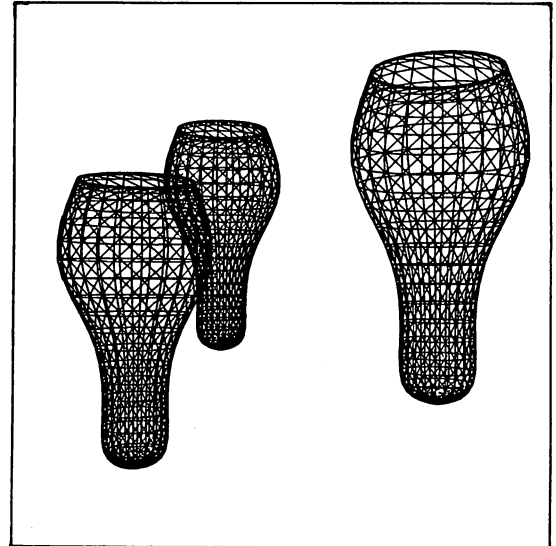


Figure 5. Scene V2

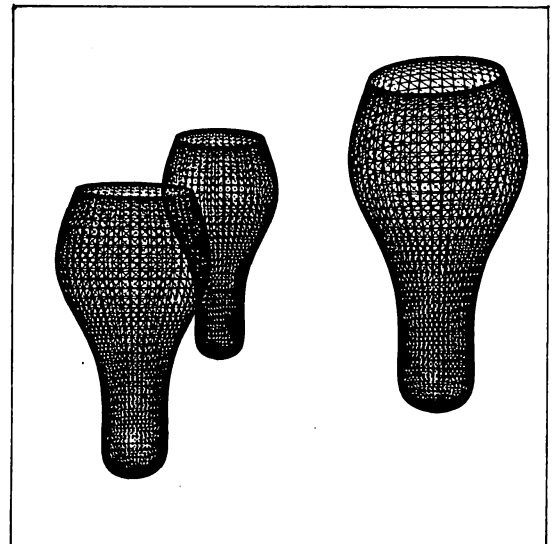


Figure 6. Scene V3

4. Hardware and operating system characteristics

All three systems were run on a *Celerity C1200* with 4MB of main memory and three 120 MB disks². The processor also has a 64K cache. The operating system was Unix† version 4.2 BSD. The three systems were compiled with the standard C compiler, using the profiling, optimization and hardware floating point options. The only difference was that RC had to use the option which prevents single floats to be cast into double, and the other two could not be run with that option. It is one more reason to be cautious about any comparison of the *absolute* times.

5. Results

Each of the seven scenes were displayed 5 times (RC, RC antialiased, DB, WS, WS antialiased). The last two scenes did not run with WS, because the allocated memory was not big enough. In the interest of brevity, we will only give two tables and three plots.

The time directly spent on I/O was not included in the tables, but is fairly constant for each system across scenes.

The first thing to note is that modelling plays no role in the first series of scenes, and the geometric transformations play little role. The dominant factors are shading and visibility determinations. In fact shading takes from 20 to more than 90% of the total time. The plots of Figures 7 and 8 show the times for visibility determination and shading for all five renderings as a function of the average polygon coverage. The plot of Figure 7 shows that the cost of visibility determination continues to climb briskly for the ray-caster even in the 5000 polygon range. It is clear from the plot of Figure 8 that DB pays the price for shading many non-visible areas as the depth complexity increases. Since all the other systems tend to flatten out as the depth complexity increases, the depth-buffer is the worse from the middle of the range explored here.

Figure 9 gives the plot of the times for visibility determination and shading for RC, RCa and DB. The main features of the statistics for the series of patch scenes are that while shading is still an important factor, the visibility determination becomes more important for the non ray-casting systems as the polygons get smaller. In fact, as expected, the RC and RCa are relatively insensitive to the size of the polygons, especially for the shading. The growth of the cost of visibility determination is less than could have been expected. In fact the ray-caster is a winner from around 5000 triangles (remember the warning against taking these absolute numbers too seriously). For the first time the cost of modelling begins to be felt, in particular for RC in scene V4. But it should be stressed that we are using fairly simple primitives here. It should also be kept in mind though that sooner or later the storage requirements will hinder RC, and they prevented us to run WS and WSa on the last two scenes.

6. Conclusions

These results are only a small sample of the profiling done so far. We tried here to define two series of scenes and choose three renderers so that the number of independent variables was relatively small. Most numbers confirmed our prejudices. Renderers using depth buffers do poorly when the depth complexity increases (here it had problems above 2) and ray-casters do well when the polygons become small. They confirmed that shading is a large part of the cost of the rendering, and it is therefore important to help with efficient routines and specialized hardware. It is important to note that DB and WS spend 10% or more of their time doing vector normalization. The computer used has hardware square root, so it was not as much a factor as it usually is in that type of programs.

The data for antialiasing is also mainly indicative. For both RC and WS antialiasing about doubles the cost and the increase comes mainly from more shading computations. We did not study here the relative cost of different illumination models and shading techniques, but we will do so as part of this study. Considering the high cost of shading, it has a significant impact on the total cost.

This brings up the issue of the quality of the picture. Of course most of these costs are assumed in the belief that a better picture will ensue. Our test pictures were as identical as we could expect, and therefore are not much help in this respect. We plan to complete a similar study with complex objects that have been modelled for other purposes with sophisticated shading and up to several hundred thousand polygons. Then the picture quality, especially as it relates to antialiasing will have to be judged subjectively.

Acknowledgements

We wish to acknowledge the support of the National Science and Engineering Research council. We also want to thank Alias Research Corporation for making available the computers and the modelling systems to create the scenes and run the profilings. John Amanatides and Tom Nadas wrote the various versions of 3D, and helped considerably in making them fit to our demands.

2. All the profiling was done on one disk 90% full.

† Unix is a trademark of AT&T Bell Laboratories

Times	RC	%	RCa	%	DB	%	WS	%	WSa	%
P1										
Total	152	100	409	100	181	100	120	100	234	100
Geometry	6	3	6	1	2	1	2	1	2	0
Visibility	81	53	292	71	5	2	11	9	34	14
Shading	50	32	96	23	167	92	103	85	194	82
P2										
Total	346	100	624	100	542	100	292	100	497	100
Geometry	12	3	12	1	2	0	2	0	2	0
Visibility	184	53	409	65	8	1	21	7	73	14
Shading	135	39	188	30	529	97	265	90	418	84
P3										
Total	614	100	794	100	964	100	365	100	645	100
Geometry	32	5	33	4	2	0	2	0	2	0
Visibility	403	65	568	71	8	0	43	11	161	24
Shading	164	26	178	22	950	98	316	86	477	73

Table 3. Some statistics for the polygon scenes

Times	RC	%	RCa	%	DB	%	WS	%	WSa	%
V1										
Total	156	100	307	100	176	100	166	100	338	100
Modelling	1	0	1	0	1	0	1	0	1	0
Geometry	6	3	6	1	0.2	0	0.2	0	0.2	0
Visibility	79	50	215	70	12	6	36	21	102	30
Shading	63	40	77	25	162	92	129	77	235	69
V2										
Total	168	100	320	100	204	100	240	100	538	100
Modelling	4	2	3	0	2	0	2	0	2	0
Geometry	10	5	9	2	0.2	0	0.2	0	0.2	0
Visibility	85	50	223	69	27	13	103	42	249	46
Shading	62	36	73	22	175	85	134	55	286	53
V3										
Total	213	100	378	100	287	100	-	-	-	-
Modelling	13	6	13	3	5.5	1	-	-	-	-
Geometry	33	15	33	8	0.2	0	-	-	-	-
Visibility	98	46	249	65	84	29	-	-	-	-
Shading	62	29	74	19	196	68	-	-	-	-
V4										
Total	376	100	591	100	574	100	-	-	-	-
Modelling	46	12	45	7	23	4	-	-	-	-
Geometry	121	32	122	20	0.2	0	-	-	-	-
Visibility	136	36	336	56	303	52	-	-	-	-
Shading	63	16	78	13	247	43	-	-	-	-

Table 4. Some statistics for the patch scenes

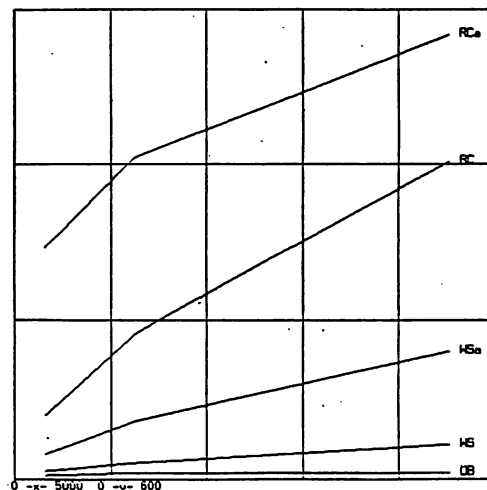


Figure 7. Visibility determination vs polygon coverage

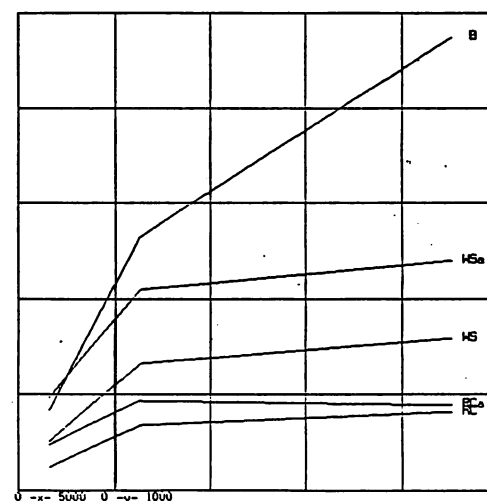


Figure 8. Shading vs polygon coverage

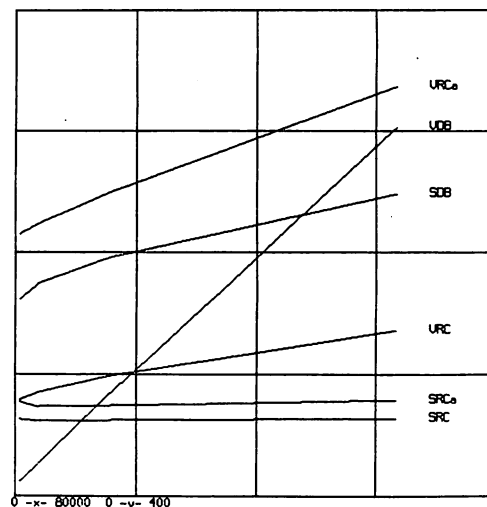


Figure 9. Visibility and shading vs subdivision

References

- [Aman84]
Amanatides, J., "Ray Tracing with Cones", in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 129-135.
- [AmBr86]
Amanatides, J. and Brown, E., *3d User's Manual*, Dynamic Graphics Project, CSRI, University of Toronto, (January 1986).
- [Cook81]
Cook, R. L. and Torrance, K. E., "A Reflectance Model for Computer Graphics", in *Proceedings of SIGGRAPH'81*, also published as *Computer Graphics*, 15, 3, (July 1981), 307-316.
- [Clar82]
Clark, J. H., "The Geometry Engine: A VLSI Geometry System for Graphics", in *Proceedings of SIGGRAPH'82*, also published as *Computer Graphics*, 16, 3, (July 1982), 127-133.
- [CoPC84]
Cook, R. L., Porter, T. and Carpenter, L., "Distributed Ray-Tracing", in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 137-145.
- [Crow81]
Crow, F. C., "A Comparison of Antialiasing Techniques", *IEEE Computer Graphics and Applications*, 1, 1, (January 1981), 40-48.
- [DiWo85]
Dippe, M. A. Z. and Wold, E. H., "Antialiasing through Stochastic Sampling", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 69-78.
- [FGHS85]
Fuchs, H., Goldfeather, J., Hultquist, J. P., Spach, S., Austin, J. D., Brooks, F. P., Eyles, J. G. and Poulton, J., "Fast Spheres, Shadows, Textures, Transparencies and Image Enhancements in Pixel-planes", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 111-120.
- [FoFC82]
Fournier, A., Fussell, D. and Carpenter, L., "Computer Rendering of Stochastic Models", *Comm. ACM*, 25, 6, (June 1982).
- [Gard84]
Gardner, G. Y., "Simulation of Natural Scenes Using Textured Quadric Surfaces", in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 11-20.
- [Gard85]
Gardner, G. Y., "Visual Simulation of Clouds", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 297-303.
- [KaGr79]
Kaplan, M. and Greenberg, D. P., "Parallel Processing Techniques for Hidden Surfaces Removal", in *Proceedings of SIGGRAPH'79*, also published as *Computer Graphics*, 13, 3, (July 1979), 300-307.
- [LeRU85]
Lee, M. E., Redner, R. A. and Useton, S. P., "Statistically Optimized Sampling for Distributing Ray Tracing", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 61-67.
- [Park80]
Parke, F. I., "Simulation and Expected Performance Analysis of Multiple Processor Z-buffer Systems", in *Proceedings of SIGGRAPH'80*, also published as *Computer Graphics*, 14, 3, (July 1980), 48-56.
- [PiFo84]
Piper, T. and A. Fournier, "A Hardware Stochastic Interpolator for Raster Displays", in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 83-91.
- [ReBl85]
Reeves, W. T. and Blau, R., "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 313-322.
- [Reev83]
Reeves, W. T., "Particle Systems- A Technique for Modeling a Class of Fuzzy Objects", in *Proceedings of SIGGRAPH'83*, also published as *Computer Graphics*, 17, 3, (July 1983), 359-376.
- [Scha83]
Schachter, B. J. (Ed.), *Computer Image Generation*, Wiley & Sons, (1983).
- [Schm81]
Schmitt, A., "Time and Space Bounds for Hidden Line and Hidden Surface Algorithms", *Proceedings of Eurographics'81*, North-Holland, (1981), 43-56.
- [SuSS74]
Sutherland, I. E., Sproull, R. F. and Schumacher, R. A., "A Characterization of Ten Hidden-Surface Algorithms", *ACM Computing Surveys*, 6, 1, (March 1974), 1-55.
- [Watk70]
Watkins, G. S., *A Real-Time Visible Surface Algorithm*, UTECH-CSC-70-101, Department of Computer Science, University of Utah, (June 1970).
- [Whel85]
Whelan, D. S., *A Multiprocessor Architecture for Real-Time Computer Animation*, Computer Science Department, California Institute of Technology, CITCS 5200:TR:85, (May 1985).
- [Whit80]
Whitted, J. T., "An Improved Illumination Model for Shaded Display", *CACM*, 26, 6, (June 1980), 343-349.

USING CACHING AND BREADTH-FIRST SEARCH TO SPEED UP RAY-TRACING

(extended abstract)

Pat Hanrahan

Abstract

Ray-tracing is an expensive image synthesis technique because many more ray-surface intersection calculations are done than are necessary to shade the visible areas of the image. This paper extends the concept of beam-tracing so that it can be coupled with caching to reduce the number of intersection tests. Two major improvements are made over existing techniques. First, the cache is organized so that cache misses are only generated when another surface is intersected, and second, the search takes place in breadth-first order so that coherent regions are completely computed before moving onto the next region.

Introduction

Ray-tracing has attracted considerable attention recently because of the super-realistic images that can be produced. Lighting and shading effects that require information about the global environment, such as shadows, reflections and refractions, can be calculated by recursively tracing rays from the surfaces they intersect [Whitted, 1980]. Distributed or stochastic ray-tracing can be used to simulate other optical effects, such as motion blur, finite-sized light sources, prismatic effects, etc., and to remove many of the artifacts due to point sampling the image [Cook, Porter and Carpenter, 1984]. Ray-casting can also be used to generate line drawings and sectioned views, and to perform the volume integrals needed for the calculation of mass properties [Roth, 1982]. Another advantage of ray-tracing is that it is conceptually elegant and easy to implement. The models that comprise the scene can be rendered if a procedure to intersect a ray with their surfaces is provided. Because of the object-oriented architecture, a ray-tracing system is easy to maintain and extend. The number of geometric primitives that can be ray-traced is quite large and continues to grow.

The major disadvantage of the standard ray-tracing algorithm is that the time needed to generate an image is equal to the number of geometric primitives *times* the size of the output image. This is because when an individual ray is being traced all the objects in the scene

need to be tested to check for an intersection. As a result there are many more ray-surface intersection calculations performed than there are rays intersecting visible surfaces. Several approaches have been attempted to reduce the number of intersection tests. Rubin and Whitted [1980] use a hierarchical tree of bounding boxes to describe a scene. Since the bounding volumes of the children lie entirely within the bounding volumes of the parent, the child volumes need only be searched if the ray intersects the parent volume. An alternative approach is to decompose space into a set of disjoint volumes. Each volume in the subdivision contains a list of those surfaces contained within it, and the subdivision is made fine enough so that the total number of objects in each volume is a small number. The search for an object intersection proceeds along the path of the ray through the subdivision. Two different subdivision methods have been reported. One method decomposes space into a rectangular array of voxels. In this case the volumes along the path of the ray can be determined using a 3-d incremental line drawing algorithm [Fujimoto and Iwata, 1985; Haeberli, 1985]. The second method decomposes space with an oct-tree [Glassner, 1984; Kaplan, 1985]. In this case, determining the next volume in the path is more complicated, but this disadvantage is offset by the fact that the oct-tree decomposition takes less space for a given level of detail.

In this paper we develop another method for speeding up the search for ray intersections by combining two methods previously reported in the literature: *beam-tracing* [Heckbert and Hanrahan, 1984] and *coherent ray-tracing* [Speer, DeRose and Barsky, 1985]. We discuss how the concept of a beam-tree can be used to characterize the coherence contained in an image. The beam-tree also suggests that the ray-surface intersections should be searched for in breadth-first order, that is, all the intersections with a given surface should be found before proceeding to the next surface. We also discuss several new methods for caching rays.

The Beam Tree

Heckbert and Hanrahan [1984] described a method to trace beams through a scene consisting of polygonal objects. This method was based on the observation that neighboring rays have essentially the same object intersection tree. This coherence can be quantified by introducing the notion of the beam-tree. In a ray-tree (as described in Whitted [1980]), the links represent rays of light and the nodes represent the surfaces that those rays intersected. Similarly, the links in a beam-tree represent beams of light, and the nodes contain a list of surfaces intersected by a beam. Each of the surfaces intersected by the beam spawns new beams corresponding to reflections, refractions and shadows.

In Heckbert and Hanrahan [1984] the beams of light were pyramidal cones. The original beam was the viewing pyramid and since the objects in the scene were polygons, all the secondary beams also had polygonal cross-sections. In the case of reflection and shadows, and under certain assumptions, refraction, it was shown that the new beams were also pyramidal cones -- that is, they contained a single apex. These restrictions allowed an entire beam to be traced at once by using a recursive polygonal hidden surface algorithm similar to that described in Weiler and Atherton [1977].

In case of curved surfaces or of true refraction, the form of a beam changes drastically when it interacts with a surface. Therefore, it is difficult to devise an algorithm to trace all the rays contained within it simultaneously. However, as we will demonstrate, it is still possible to take advantage of the coherence of a beam. In the general case we define a beam as a set of rays that all originate from the same object (or from the same point) and all intersect the same object. Generally, rays grouped together into beams will belong to adjacent pixels, although this is not strictly required. For example, all the rays through the eye that hit the background may

be considered a single beam even though the regions comprising the background may not be connected. A beam under this definition need not be uniform, for example, it might contain caustics and other singularities. An example of a coherent beam that contains a singularity is one which passes through a lense or into a crystal ball.

Notice that the beam tree for a given image (scene plus point of view) is independent of the method used to generate it. Given the ray-trees for all the pixels in the scene, the beam-tree could be created as a post-process by recursively merging adjacent rays if they intersect the same surface. The size of the beam-tree is a natural measure of the intrinsic coherence in an image.

$$\text{coherence} = \frac{\text{average ray-tree size}}{\text{total beam-tree size}}$$

Where the average ray-tree size is the total number of nodes in the ray-tree divided by the number of pixels. If at each level of the tree all the rays could be coalesced into a single beam then the size of the beam-tree would be the same as the average ray-tree size. This would be the maximum coherence possible. If the beam-tree is any bigger, this implies that adjacent ray-trees could not be merged, resulting in a new sub-beam, and therefore, less coherence. Notice that using this definition, the coherence does not depend on the relative sizes of the different sub-beams. For example, an image with one large beam and two small beams has the same coherence as an image with three equal-sized beams.

The amount of work needed to compute an image is the sum of two factors: shading and intersection processing. The number of calculations to shade the image is proportional to the size of the image times the average size of the ray tree. The optimal number of calculations needed to compute the ray-tree at each pixel is proportional to the size of the beam-tree. Optimistically, the cost of

computing each node in the beam-tree would be proportional to the number of objects in the scene. Thus, coherence allows us to decouple the complexity of the intersection phase of the calculation (which is most sensitive to the number of objects in the scene) from the shading phase of the calculation. (which is most sensitive to resolution of the image). In particular, notice that in standard ray-tracing the cost per pixel is multiplied by the number of objects, whereas using beam tracing this cost is amortized over the average number of pixels per beam. Thus, beam-tracing wins big at high resolutions.

Caching

Speer, DeRose and Barsky [1985] described a method to speed up ray-tracing which they term coherent ray-tracing. Their method is based on the same observation as contained in Heckbert and Hanrahan [1984]: that adjacent rays have a high probability of intersecting the same objects. However, instead of attempting to trace many rays simultaneously, they save the ray-tree corresponding to the previous ray and use it to guide the next intersection test. The ray-tree is intended to act as a cache. A cache hit occurs if the next ray intersects the same surface; a miss occurs if another surface is intersected. The cache is complicated by the fact that, although the same object may be hit by the next ray, another object may block the ray before it hits that object. They solved this problem by using a cache with two types of information: the last object intersected and a cylindrical region of safety. The cylinder of safety is the largest region surrounding the ray which does not contain any other surfaces. When a cache hit occurs, the ray is only tested against the last sphere and the cylinder. Thus, since only two tests need to be done, if there is a cache hit the average cost of computing an intersection per ray is constant within a beam.

We implemented this technique and upon examining caching statistics found that there were many cache misses even though the same sphere was still intersected. This is because the cylindrical region of safety is much smaller than the beam cross-section (see Figure 1). To remedy this, we devised a method which may in some situations require more work, but will cause a cache miss only if the last sphere was not intersected.

Figure 2 shows a situation where a ray hits one sphere and then hits a second sphere. The cache contains the last sphere hit and a list of spheres that could potentially block a ray travelling from the first to the second. Normally this list is empty or contains only a small number of spheres. Any ray originating on the surface of the first sphere that also intersects the second sphere can only intersect objects contained on this list of potential blocking spheres. A cache miss occurs if, first, the ray does not intersect the second sphere, or second, it intersects a sphere contained in the list of blocking spheres. Using this caching system all misses imply that a new object has been intersected. Another advantage of this technique is that no special ray-cylinder intersection tests are required.

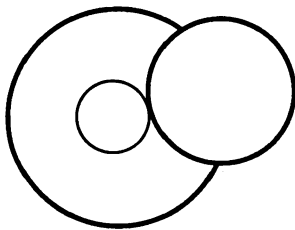


Figure 1 - This figure shows two spheres (in bold) and a cylinder of safety viewed along the axis of the ray (marked with a +). Notice that the cylinder is much smaller than the visible part of the large sphere.

There are several ways in which the list of potential blocking spheres is generated. The most common situation is where the ray originated from the eye point or is travelling to the light source. In this case the list of spheres are all those spheres contained in a cone from the point to the sphere that was intersected. The second most common situation is where a ray travels between two spheres. In this case all the spheres that lie within a cone circumscribed around the two spheres and between them are determined. Another method used to generate a list of potential blocking spheres is when a ray enters the interior of a transparent sphere and intersects itself. In this case the list of blocking spheres are all those spheres that intersect the interior of the transparent sphere. It should be mentioned that it is possible to precompute the list of potential blocking spheres for a given scene before an image is generated. However, the naive algorithm to do this is of $O(n^3)$.

This new method works well for rays that travel from sphere to sphere, from point to sphere, or from sphere to point, but does not work if a ray doesn't intersect any objects. In this case the original method due to Speer, DeRose and Barsky should be used.

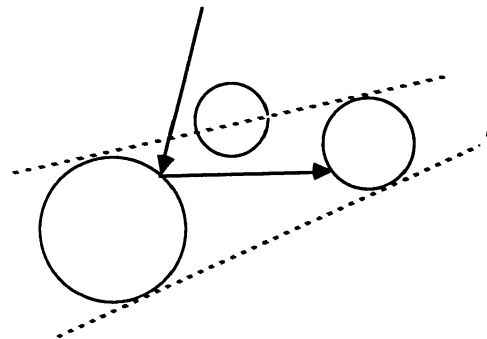


Figure 2 - This figure shows a ray reflecting off the large sphere on the left and hitting the smaller sphere on the right. Around these spheres is a cone which contains a single sphere which might potentially block another ray travelling between the same two spheres.

Breadth-first Search

Caching will work effectively if the searches are ordered in a way which maximizes the probability that a cache hit will occur. For example, if rays were randomly chosen from different pixel locations then we would expect few cache hits. Thus, the effectiveness of caching depends both on the intrinsic coherence in the scene and the search strategy employed. Fortunately, we are at liberty to reorder the search. This is analogous to the situation encountered in optimizing compilers where instruction execution order is rearranged to maximize the number of cache memory hits. The goal is to use knowledge about the general properties of the beam-tree so that searches for ray-surface intersections can be ordered in a way that maximizes the probability of cache hits.

The first important point is that the cache should be organized as a tree. There is little reason to suspect that a reflected ray will hit the same object as the refracted ray or the incident ray. Practically this means that if a cache miss occurs at a parent node then we should flush the cache of all the child nodes.

The second and major point is that the beam's cross-section is two-dimensional, not one-dimensional. Consider the simplified case where the ray-tracer is only being used to remove hidden surfaces, so that there are no reflected or transmitted rays. In the standard ray-tracer, rays are generated in scanline order. The number of cache misses per scanline is equal to the number of regions crossing that scanline. Each cache miss causes all the objects to be searched. The total number of complete searches is therefore much greater than the total number of regions. In order to achieve one complete search per region the search should continue two-dimensionally until a cache miss occurs. This is similar to the common seed fill or boundary fill algorithm used in paint systems [Smith, 1979]. This search method also works when the tree has greater depth. If we imagine

the complete ray-tree as including all the rays emanating from the eye point, the region fill corresponds to a breadth-first search of this tree.

The initial reaction to breadth-first search is that since images are so large, the size of the list of rays queued would be prohibitively large. However, it is possible to organize the search in scanline order so that the list is kept to a reasonable size.

Results

To test these ideas we implemented a simple cached ray-tracer for spheres. The code was written so it was easy to turn caching on or off. This program was run over a variety of different scenes with similar results. In the table below the scene consisted of a $N \times N \times N$ array of spheres whose centers and radii were randomly jittered. This cube of spheres was then viewed from an angle and scaled so that it filled the screen. As can be seen from Table 1, caching itself sped up the ray-tracer from 2-5 times; adding breadth-first search sped it up by another factor of 2-3.

Number of spheres	3^3	4^3	5^3
Not cached	1.00	1.00	1.00
Cached	.53	.29	.20
Cached, breadth-first	.25	.15	.11

Table 1 - Timing results

Conclusions

Although these results are preliminary, this area of research looks promising. Analyzing the caching statistics and comparing them to the actual coherence as measured by the beam-tree we find that the number of complete searches is still much more than the theoretical maximum. In particular the cache hit test is not very

successful when a ray had not previously hit an object. Improving this case would significantly speed up the program.

Spheres were chosen in this study because the intersection tests are easy to implement and because spheres can also be used to bound the extent of other object types. An avenue for further research is to devise cache tests for other object types. In the general case it may be worthwhile to allow different caching strategies for different objects. For example, the polygons in a convex polyhedral solid cannot occlude other polygons of that solid. Thus, these polygons cannot be on the list of potential blockers. Sometimes even in the case when a search through all the objects in the scene need be done, a cache can be used to speed this up. Considering again the case of a convex polyhedral solid, we can cache the last polygon hit. If that polygon is missed by the next ray, then we should search polygons adjacent to it first.

Caching is a very general method for speeding up computations when coherence exists. For this reason it can be used along with other methods, such as cellular decomposition, to speed up the search for ray-surface intersections. It is also likely that many other hidden surface algorithms would benefit from caching.

Finally, the idea of breadth-first search was originally motivated by the desire to build an interactive ray-tracing tool. If each ray is immediately painted onto the image after it is traced, then the details of the image will gradually be filled in. First, the a hidden surface view will be drawn, followed by reflections and shadows at greater and greater depth.

Acknowledgements

Some of this work was done while the author was at the University of Wisconsin and a member of the research staff of the Digital Equipment Corporations Systems Research Center. Thanks to Paul Haeberli for his insight and enthusiasm.

References

Cook, R., Porter, T. and Carpenter, L., Distributed ray tracing, *Computer Graphics*, (SIGGRAPH '84 Proceedings) 18(3), pp. 137-145, 1984.

Fujimoto, A., and Iwata, K., Accelerated ray-tracing, *Computer Graphics*, Tokyo, 1985, pp. 1-26, 1985.

Glassner, A.S., Octree encoding to speed up ray tracing, *IEEE Trans. Comp. Graphics and Appl.*, 4(10), pp. 15-22, 1984.

Haeberli, P., (personal communication) 1985.

Heckbert, P.S. and Hanrahan, P., Beam tracing polygonal objects, *Computer Graphics* (SIGGRAPH '84 Proceedings), 18(3), pp. 119-127, 1984.

Kaplan, M., Constant-time ray tracing, Notes for State of the Art in Image Synthesis, Proc. of SIGGRAPH '85, 1985.

Roth, S.D., Ray casting for modeling solids, *Computer Graphics and Image Processing*, 18(2), pp. 109-144, 1982.

Rubin, S.M., and Whitted, T., A 3-dimensional representation for fast rendering of complex scenes, *Computer Graphics* (SIGGRAPH '80 Proceedings), 14(3), pp. 110-116, 1980.

Smith, A.R., Tint fill, *Computer Graphics* (SIGGRAPH '79 Proceedings) 13(2), pp. 276-283, 1979.

Speer, L.R., DeRose, T.D., and Barsky, B.A., A theoretical and empirical analysis of coherent ray-tracing, *Graphics Interface '85*, 1985.

Weiler, K.J., and Atherton, P.A., Hidden surface removal using polygon area sorting, *Computer Graphics* (SIGGRAPH '77 Proceedings), 11(3), 1977.

Whitted, T., An improved illumination model for shaded display, *C.A.C.M.*, 23(6), pp. 343-349, 1980.

What are Visual Programming, Programming by Example, and Program Visualization?

Brad A. Myers

Dynamic Graphics Project
Computer Systems Research Institute
University of Toronto
Toronto, Ontario, M5S 1A4
Canada

ABSTRACT

There has been a great interest recently in systems that use graphics to aid in the programming, debugging, and understanding of computer programs. The terms "Visual Programming" and "Program Visualization" have been applied to these systems. Also, there has been a renewed interest in using examples to help alleviate the complexity of programming. This technique is called "Programming by Example." This paper attempts to provide more meaning to these terms by giving precise definitions, and then uses these definitions to classify existing systems into a taxonomy.

RESUME

Les systèmes qui utilisent l'infographie pour aider à la programmation, à la mise-au-point et à la compréhension de logiciels ont récemment suscité beaucoup d'intérêt. Les termes "programmation visuelle" et "visualisation de programmes" ont été associés à ces systèmes. Il y a aussi eu un renouveau d'intérêt pour l'utilisation d'exemples pour aider à simplifier la programmation. On parle alors de "programmation par exemples". Nous essaierons de définir ces termes avec plus de précision et utiliserons ces définitions comme base pour établir une taxonomie des systèmes disponibles actuellement.

Key Words and Phrases: Visual Programming, Program Visualization, Programming by Example, Inferencing, Automatic Programming, Flowcharts, Debugging Aids, Program Synthesis, Documentation, Computer Languages.

Extended Summary.

NOTE: This paper is a summary of [Myers 86]. The reader should refer to that paper for full information.

As the distribution of personal computers and the more powerful personal workstations grows, the majority of computer users now do not know how to program. They buy computers with packaged software and are not able to modify the software even to make small changes. In order to allow the end user to reconfigure and modify the system, the software may provide various options, but these often make the system more complex and still may not address the users' problems. "Easy-to-use" software, such as the "Direct Manipulation" systems [Shneiderman 83] actually make the user-programmer gap *worse* since more people will be able to use the software (since it is easy to use), but the internal program code is now much more complicated (due to the extra code to handle the user interface). Therefore, systems are moving in the direction of providing end user programming. It is well-known that conventional programming languages are difficult to learn and use [Gould 84], requiring skills that many people do not have. In an attempt to make the programming task easier, recent research has been directed towards using graphics. This has been called "Visual Programming" or "Graphical Programming". Some Visual Programming systems have successfully demonstrated that non-programmers can create fairly complex programs with little training [Halbert 84].

Another motivation for using graphics is that it tends to be a higher-level description of the desired actions (often de-emphasizing issues of syntax and providing a higher level of abstraction) and may therefore make the programming task easier even for professional programmers. This may be especially true during debugging, where graphics can be used to present much more information about the program state (such as current variables and data structures) than is possible with purely textual displays. This is one of the goals of Program Visualization. Other Program Visualization systems use graphics to help teach computer programming.

Programming-by-Example is another technology that has been investigated to make programming easier, especially for non-programmers. It involves presenting to the computer examples of the data that the program is supposed to process and using these examples during the development of the program. Many, although not all, Programming-by-Example systems have also used Visual Programming, so these two technologies are often linked.

Recently, there has been a large number of articles about systems that incorporate some or all of these features [Grafton 85][Raeder 85]. Unfortunately, the terms have been used imprecisely¹, and there has not been a comprehensive taxonomy that classifies these systems. This paper summarizes research that attempts fill this gap in the literature. The full results are reported in [Myers 86]. First, the important terms are defined in a precise manner, and then these definitions are used to differentiate some example systems.

There are many systems that could be included in this paper in the various categories, but no attempt has been made to be comprehensive. It is hoped that the selection of systems listed will help the reader understand the intent of the classification system.

Definitions.

Programming. What is meant by computer "programming" is probably well understood, but it is important to have a definition that can be used to eliminate some limited systems. In this paper, "program" is defined as "a set of statements that can be submitted as a unit to some computer system and used to direct the behavior of that system" [Oxford 83]. While the ability to compute "everything" is not required, the system must include the ability to handle conditionals and iteration, at least implicitly.

Interactive vs. Batch. Any programming language system may either be "interactive" or "batch." A batch system has a large processing delay before statements can be run while they are compiled, whereas an interactive system allows statements to be executed when they are entered. This characterization is actually more of a continuum than a dichotomy since even interactive languages like LISP typically require groups of statements (such as an entire procedure) to be specified before they are executed.

¹For example, Zloof's Query-By-Example system [Zloof 77 and 81] is not a Programming by Example system.

Visual Programming. "Visual Programming" (VP) refers to any system that allows the user to *specify* a program in a two (or more) dimensional fashion. Conventional textual languages are not considered two dimensional since the compiler or interpreter processes it as a long, one-dimensional stream. Visual Programming includes conventional flow charts and graphical programming languages. It does not include systems that use conventional (linear) programming languages to define pictures. This eliminates most graphics editors, like Sketchpad [Sutherland 63].

Program Visualization. "Program Visualization" (PV) is an entirely different concept from Visual Programming. In Visual Programming, the graphics is the program itself, but in Program Visualization, the program is specified in the conventional, textual manner, and the graphics is used to illustrate some aspect of the program or its run-time execution. Unfortunately, in the past, many Program Visualization system have been incorrectly labeled "Visual Programming" (as in [Grafton 85]). Program Visualization systems can be divided along two axes: whether they illustrate the *code* or the *data* of the program, and whether they are *dynamic* or *static*. "Dynamic" refers to systems that can show an animation of the program running, whereas "static" systems are limited to snapshots of the program at certain points. If a program created using Visual Programming is to be displayed or debugged, clearly this should be done in a graphical manner, but this would not be considered Program Visualization. Although these two terms are similar and confusing, they have been widely used in the literature, so it was felt appropriate to continue to use the common terms.

Programming by Example. The term "Programming by Example" (PBE) has been used to describe a large variety of systems. Some early systems attempted to create an entire program from a set of input-output pairs. Other systems require the user to "work through" an algorithm on a number of examples and then the system tries to *infer* the general program structure. This is often called "automatic programming" and has generally been an area of Artificial Intelligence research.

Recently, there have been a number of systems that require the user to specify everything about the program (there is no inference involved), but the user can work out the program on a specific example. The system executes the user's commands normally, but remembers them for later re-use. Bill Buxton coined the phrase "Programming *with* Examples" to more accurately describe these systems. Halbert [84] characterizes Programming with Examples as "Do What I Did" whereas inferential Programming by Example might be "Do What I Mean". The term "Programming by Example" will be used to include both inferring systems and Programming With Example systems.

Of course, whenever code is executed in any system, test data must be entered to run it on. The distinction between normal testing and "Programming with Examples" is that in the latter the system requires or encourages the specification of the examples *before* programming begins, and then applies the program as it develops to the examples. This essentially requires all Programming-with-Example systems (but not Programming-by-Example systems with inferencing) to be interactive.

Taxonomy of Programming Systems.

This paper presents two taxonomies. The first is for systems that support programming. The second taxonomy is for systems that use graphics *after* the programming process is finished (Program Visualization systems).

A meaningful taxonomy can be created by classifying *programming systems* into eight categories using the orthogonal criteria of

- Visual Programming or not,
- Programming by Example or not, and
- Interactive or batch.

Of course, a single system may have features that fit into various categories and some systems may be hard to classify, so this paper attempts to characterize the systems by their most prominent features. Figure 1 shows the division with some sample systems.

Taxonomy of Program Visualization Systems.

The systems listed below are not *programming* systems since code is created in the conventional manner. Graphics in these are used to *illustrate* some aspect of the program after it is written. Figure 2 shows some Program Visualization systems classified by whether they attempt to illustrate the code or the data of a program (some provide both), and whether the displays are static or dynamic.

Conclusions.

Visual Programming, Programming by Example and Program Visualization are all exciting areas of active computer science research, and they promise to improve the user interface to programming environments. A number of interesting systems have been created in each area, and there are some that cross the boundaries. This paper has attempted to classify some of these systems in hopes that this will clarify the use of the terms and provide a context for future research.

ACKNOWLEDGEMENTS

For help and support of this article, I would like to thank Bill Buxton, Ron Baecker, Bernita Myers, and many others at the University of Toronto. The research described in this paper was partially funded by the National Science and Engineering Research Council (NSERC) of Canada.

Not Programming by Example

	Batch	Interactive
Not VP	All Conventional Languages: Pascal, Fortran, etc.	LISP, APL, etc.
VP	Grail [Ellis 69] AMBIT/G/L [Christensen 68,71] Query by Example [Zloof 77, 81] FORMAL [Shu 85] GAL [Albizuri-Romero 84]	Graphical Program Editor [Sutherland 66] PIGS [Pong 83] Pict [Glinert 84] PROGRAPH [Pietrzykowski 83,84] State Transition UIMS [Jacob 85]

Programming by Example

	Batch	Interactive
Not VP	I/O pairs* [Shaw 75]	Tinker [Lieberman 82]
VP	[Bauer 78] traces*	AutoProgrammer* [Biermann 76] Pygmalion [Smith 77] Graphical Thinglab [Borning 86] SmallStar [Halbert 81,84] Rehearsal World [Gould 84]

Figure 1.
Classification of programming systems by whether they are visual or not, whether they have Programming by Example or not, and whether they are interactive or batch. Starred systems (*) have inferencing, and non-starred PBE systems use Programming With Example.

	Static	Dynamic
Code	Flowcharts [Haibt 59] SEE Visual Compiler [Baecker 86] PegaSys [Moriconi 85]	BALSA [Brown 84] PV Prototype [Brown 85]
Data	TX2 Display Files [Baecker 68] Incense [Myers 80,83]	Two Systems [Baecker 75] Sorting out Sorting [Baecker 81] BALSA [Brown 84] Animation Kit [London 85] PV Prototype [Brown 85]

Figure 2.
Classification of Program Visualization Systems by whether they illustrate code or data, and whether they are dynamic or static.

REFERENCES

- [Albizuri-Romero 84] Miren B. Albizuri-Romero. "GRASE--A Graphical Syntax-Directed Editor for Structured Programming," *SIGPLAN Notices*. 19(2) Feb. 1984. pp. 28-37.
- [Attardi 82] Giuseppe Attardi and Maria Simi. "Extending the Power of Programming by Example," *SIOGA Conference on Office Information Systems*, Philadelphia, PA, Jun. 21-23, 1982. pp. 52-66.
- [Baecker 68] R.M.Baecker. "Experiments in On-Line Graphical Debugging: The Interrogation of Complex Data Structures," (Summary only) *First Hawaii International Conference on the System Sciences*. Jan. 1968. pp. 128-129.
- [Baecker 75] R.M.Baecker. "Two Systems which Produce Animated Animated Representations of the Execution of Computer Programs," *SIGCSE Bulletin*. 7(1) Feb. 1975. pp. 158-167.
- [Baecker 81] Ron Baecker. *Sorting out Sorting*. 16mm color, sound film, 25 minutes. Dynamics Graphics Project, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada. 1981. Presented at ACM SIGGRAPH'81. Dallas, TX. Aug. 1981.
- [Baecker 86] Ronald Baecker and Aaron Marcus. "Design Principles for the Enhanced Presentation of Computer Program Source Text," *Human Factors in Computing Systems: Proceedings SIGCHI'86*. Boston, MA. Apr. 13-17, 1986.
- [Bauer 78] Michael A. Bauer. *A Basis for the Acquisition of Procedures*. PhD Thesis, Department of Computer Science, University of Toronto. 1978. 310 pages.
- [Biermann 76] Alan W. Biermann and Ramachandran Krishnaswamy. "Constructing Programs from Example Computations," *IEEE Transactions on Software Engineering*. SE-2(3) Sept. 1976. pp. 141-153.
- [Borning 86] Alan Borning. "Defining Constraints Graphically," *Human Factors in Computing Systems: Proceedings SIGCHI'86*. Boston, MA. Apr. 13-17, 1986.
- [Brown 84] Marc H. Brown and Robert Sedgewick. "A System for Algorithm Animation," *Computer Graphics: SIGGRAPH'84 Conference Proceedings*. Minneapolis, Minn. 18(3) July 23-27, 1984. pp. 177-186.
- [Brown 85] Gretchen P. Brown, Richard T. Carling, Christopher F. Herot, David A. Kramlich, and Paul Souza. "Program Visualization: Graphical Support for Software Development," *IEEE Computer*. 18(8) Aug. 1985. pp. 27-35.
- [Christensen 68] Carlos Christensen. "An Example of the Manipulation of Directed Graphs in the AMBIT/G Programming Language," in *Interactive Systems for Experimental Applied Mathematics*, Melvin Klerer and Juris Reinfelds, eds. New York: Academic Press, 1968. pp. 423-435.
- [Christensen 71] Carlos Christensen. "An Introduction to AMBIT/L, A Diagrammatic Language for List Processing," *Proc. 2nd Symposium on Symbolic and Algebraic Manipulation*. Los Angeles, CA. Mar. 23-25, 1971. pp. 248-260.
- [Ellis 69] T.O. Ellis, J.F. Heafner and W.L. Sibley. *The Grail Project: An Experiment in Man-Machine Communication*. RAND Report RM-5999-Arpa. 1969.
- [Glinert 84] Ephraim P. Glinert and Steven L. Tanimoto. "Pict: An Interactive Graphical Programming Environment," *IEEE Computer*. 17(11) Nov. 1984. pp. 7-25.
- [Gould 84] Laura Gould and William Finzer. *Programming by Rehearsal*. Xerox PARC TR SCL-84-1. May, 1984. 133 pages. Excerpted in *Byte*. 9(6) June, 1984.
- [Grafton 85] Robert B. Grafton and Tadao Ichikawa, eds. *IEEE Computer*, Special Issue on Visual Programming. 18(8) Aug. 1985.
- [Haibt 59] Lois M. Haibt. "A Program to Draw Multi-Level Flow Charts," *Proceedings of the Western Joint Computer Conference*. San Francisco, CA. 15 Mar. 3-5, 1959. pp. 131-137.
- [Halbert 81] Daniel C. Halbert. *An Example of Programming by Example*. Masters of Science Thesis. Dept. of EE&CS, University of Calif., Berkeley and Xerox Corporation Office Products Division, Palo Alto, CA. June, 1981. 55 pages.
- [Halbert 84] Daniel C. Halbert. *Programming by Example*. PhD Thesis. Computer Science Division, Dept. of EE&CS, University of California, Berkeley. 1984. Also: Xerox Office Systems Division, Systems Development Department, TR OSD-T8402, December, 1984. 83 pages.
- [Jacob 85] Robert J.K. Jacob. "A State Transition Diagram Language for Visual Programming," *IEEE Computer*. 18(8) Aug. 1985. pp. 51-59.
- [Lieberman 82] Henry Lieberman. "Constructing Graphical User Interfaces by Example," *Graphics Interface'82*, Toronto, Ont. Mar. 17-21, 1982. pp. 295-302.
- [London 85] Ralph L. London and Robert A. Drusberg. "Animating Programs in Smalltalk," *IEEE Computer*. 18(8) Aug. 1985. pp. 61-71.
- [Moriconi 85] Mark Moriconi and Dwight F. Hare. "Visualizing Program Designs Through PegaSys," *IEEE Computer*. 18(8) Aug. 1985. pp. 72-85.
- [Myers 80] Brad A. Myers. *Displaying Data Structures for Interactive Debugging*. Xerox Palo Alto Research Center Technical Report CSL-80-7. June, 1980. 97 pages.
- [Myers 83] Brad A. Myers. "Incense: A System for Displaying Data Structures," *Computer Graphics: SIGGRAPH '83 Conference Proceedings*. 17(3) July 1983. pp. 115-125.
- [Myers 86] Brad A. Myers. "Visual Programming, Programming by Example, and Program Visualization; A Taxonomy," *Proceedings SIGCHI'86: Human Factors in Computing Systems*. Boston, MA. April 13-17, 1986.
- [Oxford 83] *Dictionary of Computing*. Oxford: Oxford University Press, 1983.
- [Pietrzykowski 83] Thomas Pietrzykowski, Stanislaw Matwin, and Tomasz Muldner. "The Programming Language PROGRAPH: Yet Another Application of Graphics," *Graphics Interface'83*, Edmonton, Alberta. May 9-13, 1983. pp. 143-145.
- [Pietrzykowski 84] T. Pietrzykowski and S. Matwin. *PROGRAPH: A Preliminary Report*. University of Ottawa Technical Report TR-84-07. April, 1984. 91 pages.
- [Pong 83] M.C. Pong and N. Ng. "Pigs--A System for programming with Interactive Graphical Support," *Software--Practice and Experience*. 13(9) Sept. 1983. pp. 847-855.
- [Raeder 85] Georg Raeder. "A Survey of Current Graphical Programming Techniques," *IEEE Computer*. 18(8) Aug. 1985. pp. 11-25.
- [Shaw 75] David E. Shaw, William R. Swartout, and C. Cordell Green. "Inferring Lisp Programs from Examples," *Fourth International Joint Conference on Artificial Intelligence*. Tbilisi, USSR. Sept. 3-8, 1975. 1 pp. 260-267.
- [Shneiderman 83] Ben Shneiderman. "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer*. 16(8) Aug. 1983. pp. 57-69.
- [Shu 85] Nan C. Shu. "FORMAL: A Forms-Oriented Visual-Directed Application Development System," *IEEE Computer*. 18(8) Aug. 1985. pp. 38-49.
- [Smith 77] David C. Smith. *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*. Basel, Stuttgart: Birkhauser, 1977. 187 pages.
- [Sutherland 63] Ivan E. Sutherland. "SketchPad: A Man-Machine Graphical Communication System," *AFIPS Spring Joint Computer Conference*. 23 1963. pp. 329-346.
- [Sutherland 66] William R. Sutherland. *On-line Graphical Specification of Computer Procedures*. MIT PhD Thesis. Lincoln Labs Report TR-405. 1966.
- [Zloof 77] Moshe M. Zloof and S. Peter de Jong. "The System for Business Automation (SBA): Programming Language," *CACM*. 20(6) June, 1977. pp. 385-396.
- [Zloof 81] Moshe M. Zloof. "QBE/OBE: A Language for Office and Business Automation," *IEEE Computer*. 14(5) May, 1981. pp. 13-22.

An Editing Model for Generating Graphical User Interfaces

by
Dan R. Olsen Jr.

Computer Science Department
Brigham Young University

Abstract

A basic architecture for a User Interface Management System is presented. The problem of updating a display in response to interactive commands is discussed. The basic architecture is then extended to include basic editing and browsing processes on arbitrary data structures. Editing templates are presented as a technique which embodies the entire manipulation process for a particular data structure / data display combination. Such templates in conjunction with the User Interface Management System are able to automatically provide a majority of the code required in an interactive application.

Introduction

Within the graphics community the concept of a User Interface Management System (UIMS) has come into usage[THO83]. Such systems have been developed to overcome the high cost of implementing interactive graphics programs with quality human-computer interfaces. Most such systems have concentrated on the problem of input dialogue management [KAM83, GRE85, JAC83, VAN83]. Having developed three such systems in our laboratory[OLS83, OLS85a, OLS85b, OLS85c], we have become concerned with the problem of data display in an interactive program. In implementing interactive programs we have found that the input dialogue can be programmed in a matter of hours or days, using our tools, but the code to update the display after each modification to the application data structure takes months to implement.

In attacking this display update problem we approached it from the point of view of an intelligent display processor. Our original architecture for an interactive program is shown in Figure 1.

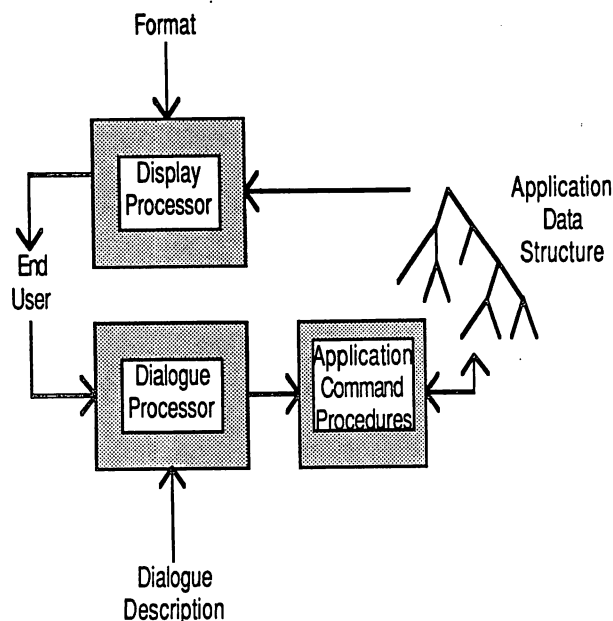


Figure 1.

In this architecture the input events are parsed according to the dialogue description and, based on the input, one or more of the application command procedures is invoked. It is the responsibility of these application command procedures to update some application data structure. It is the role of the display processor to create a graphical presentation of the application data according to the specified format. There are a wide range of possible formats for displaying the data which will only lightly be touched upon here. This architecture is somewhat similar to that proposed at the Seeheim workshop on user interface management [PFA85]. The key problem of interest in this paper is how the graphical image should be updated whenever the application data is changed. The obvious solution is to simply redraw the entire image. This is an extremely poor solution if one desires reasonable response time.

After some experience with the above model we determined that a closer relationship between the data editing commands and the display update functions is essential. A more acceptable system architecture is shown in Figure 2.

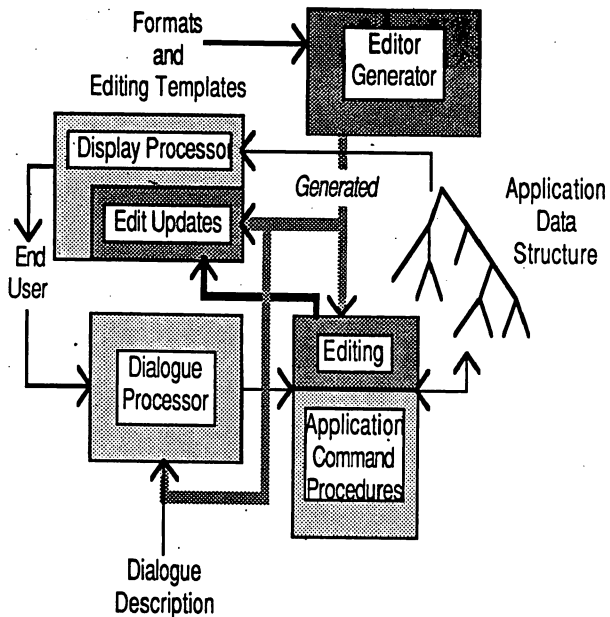


Figure 2.

In this architecture a large portion of the interactive dialogue is viewed as a process of browsing and/or editing the application data structure. Based on this view, a description of the application data is taken together with a selection of browsing and editing methods to generate the dialogue definition, display update procedures and application command procedures for the selected operations. Note that this does not generate the entire interactive program but simply the editing portion which is the most display update intensive. Any analysis, file management or help facilities can all be added. In our experience it is the editing and browsing operations with dominate an interactive program.

This paper will then proceed in the following fashion. First a data management facility called STUF (STructured Files) will be reviewed so as to define the possible data structures that might be represented in this UIMS model. This will be followed by a short description of MIKE (Menu Interaction Kontrol Environment) which is our input dialogue system. Having set the stage for our editing environment, editing templates will be described which provide the screen update facilities that we are interested in.

STUF

The STUF package was developed as a data model specifically for interactive applications. Our desire was to create a data model which would behave equally well in main memory or on secondary storage. We also imposed the requirement that STUF data must be accessible either relationally (as in a relational database) or as a linked structure (as with pointers in Pascal). The reason for this is that relational data models are very powerful but in many cases are too inefficient for practical graphics use.

Each STUF file has a specific *filetype* which is defined by a set of *datatypes*. A datatype is defined to be either a Record or a Union. A datatype consists of a list of fields each of which has a *fieldtype*. A Record is defined to have all of the fields in the list. A Union is defined to have one of the fields in the list (as determined by a tag field at run time). A Union is similar in capability to a Pascal variant record. A fieldtype is either an Integer, Real, Char, Boolean or a reference to an object of

some other datatype. A Field can also have dimension so as to create fixed length arrays.

The links or references to other datatype objects pose a problem. In a relational model, tuples are linked to each other by matching key values. In programming languages this linkage is handled with pointers. The pointer solution provides the efficiency that we require but when pointers are written to secondary storage they lose all meaning. In addition the pointer model does not provide the flexibility found in the relational model. The STUF solution is to create a variable length array of tuples for each datatype in the file. A reference to a tuple is then stored as an index into this array. New and Dispose procedures are provided to allocate and deallocate tuples within a data type. In addition, the undeleted tuples in one of the arrays can be treated as a relation in the sense of a relational database to provide an associative access method for tuples. Since tuple indices do not lose their meaning when saved to disk or passed to another program we have our needed data facility.

One more point to consider is that of cursors for editing. When one is editing it is usually performed by moving some cursor or *current data object pointer*. Editing and browsing operations are then performed relative to this current data object pointer. When editing a STUF data file, such a cursor is represented as a tuple index. Many of the editing templates described below will be defined in terms of such cursors.

MIKE

MIKE is our dialogue handling system which is described in more detail elsewhere [OLS85c]. The important point to understand is how MIKE models the input dialogue. The basic interactive unit in MIKE is the command. A MIKE command is simply a Pascal procedure or function. MIKE accepts as its initial dialogue description a set of such procedures and then generates a compiled interface to these procedures. Interactively all procedures are presented in a menu and the selected menu item becomes the current command. Having selected a command, MIKE then prompts with a menu of all functions whose result type is the same as the type of the command's first parameter. These types can be any type from the application program. In addition to the functions in the menu, MIKE can supply primitive inputs for integer, real, string, function key and point types. MIKE continues accepting inputs until a complete command expression has been parsed. The command expression syntax is very similar to Pascal procedure invocation syntax with the exception that no punctuation, such as commas and parenthesis, is actually input. MIKE's interactive use of functions and procedures is similar in many ways to Smalltalk's interactive use of methods [GOL83].

Given such a primitive interface, a profile editor can be used to improve it. The profile editor allows menus to be restructured, prompts and echos changed, icons drawn, function buttons mapped to commands and help texts written. This fleshes out and enhances the user's view of the interface but it does not change the underlying model of command procedures and functions. The profile editor serves a similar role to Buxton's MenuLay [BUX83] in editing external presentations of dialogues.

This command model has proven to be much simpler to use than the state machine and grammar approaches that we have used previously. For the purposes of our editing model of interaction we can characterize all changes to the application data structure as Pascal procedures or functions. That is for each kind of change to be made to the data structure we generate a procedure which makes the change and performs

any necessary display updates. The generator then informs MIKE of the procedures' names and parameter types. From this information MIKE can create the necessary user input dialogue interface.

Editing Templates

The editor generation concept is based on the idea of editing templates. An editing template is designed as a presentation of a particular general class of data structures. Linked lists, symbol tables and trees are examples of such structure classes. An editing template then consists of a set of routines to do the following:

- display application data from a STUF file using the model;
- provide data structure traversal commands for browsing through the data image being displayed by the model, and
- provide the editing commands for creating, deleting and/or modifying the data presented using the model.

It should be noted that the commands provided must, in addition to performing their intended tasks, also update the display to reflect the results of their tasks. It is the display update which is most important to our discussion here.

In addition to the services that an editing template provides, each template also has a set of parameters which are used when creating an instance of a template. In this sense an editing template can be thought of as a macro except that the parameters may be lists and other data structures rather than simple text to be substituted.

In Smalltalk an editing template would be defined as a class. The services that it provides would be methods of the class and the parameters would be methods of the objects that the template is manipulating. In ADA an editing template could in most cases be represented as a generic package[GEH84]. Our work has been done in Pascal using a macro preprocessor but the concepts are the same. An editing template then is characterized by the data structure that it represents. An example is given below of an editing template for linked lists.

Linked-List Editing Template

As a first example of how editing templates function a linked list is appropriate. Since an editing template is meant to be a generic capability it must know a number of things about the linked list that it is to display. For example it must know how to find the head of the list, what field is used as the link for the list, how to display one of the elements of the list and how much display space to allocate to each element. This kind of information is provided by the following set of parameters. These parameters are all preceded by a percent sign so that their text is easily recognized in the generated routines.

- %ObjType - The data type of the list elements to be displayed.
- %Link - The field that is used to link elements of ObjType together.
- %CurObj - The tuple index variable that is to be used as the cursor in moving up and down the linked list. This is an index into ObjType's array.
- %HeadType - The type of object where the head of the list is stored.
- %HeadField - The field in HeadType that points to the head of the linked lists being displayed. This field must reference data type ObjType.

- %CurHeadObj - The tuple index variable that references the HeadType tuple which contains the head of the current list being displayed.
- %XExt and %YExt - These contain the X and Y sizes of the screen space to be allocated to each displayed element.
- %Window - This is the number of the window that the list is to be displayed in.
- %ObjDisp(ObjIndex; X,Y) - This is a routine to be called to have the list element referenced by ObjIndex displayed on the screen at location (X,Y).
- %ObjDel(ObjIndex) - This is a routine which will clean up and delete the list element reference by ObjIndex.
- %EditSpace - This is the number of list element spaces to be left empty on the screen so that new elements can be inserted into the list without necessitating a repainting of the entire list.

The various interactive tasks that one would want to perform on the display of a linked list would include moving a cursor up and down the list (scrolling the list if necessary), inserting a new element in the list and deleting the current element from the list. In addition to these primitive operations one may also want to select a current item from the list using some other criteria such as a name. To accomplish all of these the template would provide the following command procedures directly to MIKE.

- %WindowUp
{ move the list cursor up one element scrolling if necessary }
- %WindowDown;
{ move the list cursor down one element scrolling if necessary }
- %WindowLeft and %WindowRight
{ if, because of the size and shape of the window, the list is displayed in multiple columns then this will move the cursor left or right as the case may be and update %CurObj appropriately }
- %WindowPageUp and %WindowPageDown
{ if the list is longer than will fit in the window then this will move the cursor backward and forward through the list one page at a time }
- %WindowDelete;
{ delete the current object pointed at by %CurObj }

In addition to these command procedures which are exposed to MIKE the following additional service routines are generated.

- Restore%Window
{ This completely refreshes the window whenever the window itself changes or the value of %CurHeadObj changes. }
- %WindowUpdateCur;
{ This will simply update the display of the current object due to some modification to it by some other command }
- %WindowInsert(ObjIdx)
{ This will insert the specified object into the linked list immediately after %CurObj and make it the current object updating the display appropriately }
- %WindowChangeCursor(ObjIdx)
{ This will change %CurObj to the value of ObjIdx and update the screen appropriately }

Note that all of the names of the generated routines are parameterized by %Window so as to make them unique. Note

also that there is no insert command exposed directly to MIKE because of the variety of ways that an application may want to create and initialize elements of the list. After such an element is created by some command procedure the `%WindowInsert` service procedure can be called which will handle the list insertion and screen update tasks properly.

Note that the actual display of an element is left up to an application supplied procedure. In many cases we have had lists of lists to display. This is handled with two windows, `%W1` and `%W2` for example. `%W1` displays the main list and can be browsed and edited as shown above. `%W1`'s `%CurObj` cursor is also the `%CurHeadObj` for `%W2`. The `%ObjDisp` procedure for `%W1` contains a call to `Restore%W2` so that any change of the cursor in `%W1` will cause an update of `%W2`. By combining the linked list template with the various other templates a large number of data display and manipulation techniques can be implemented very quickly.

Other possibilities

The set of services provided above is not necessarily complete nor the only possible approach. For example the linked-list template could have a clipboard variable added as a parameter and then supply to MIKE the necessary commands to Cut, Paste and Copy list segments to and from the window's clipboard. An additional feature might be an element from the list with a mouse rather than scrolling through the list.

Other data structuring mechanisms that we have implemented include unordered and sorted associative lists. Such lists are accessed by name or some other criteria. These two techniques view a given STUF datatype as a table of tuples for a relation and allow scrolling through and editing of such tables based on application supplied sort orders and selection criteria.

These structure editing templates are being combined with a forms editor which handles the element display functions. Other templates which we are still working on would handle network or schematic type displays such as is shown in Figure 3.

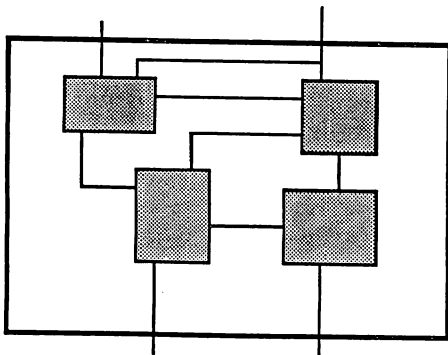


Figure 3.

Such display forms are more difficult because of the layout connection routing aids that one would want to provide automatically.

Summary

Our User Interface Management System therefore consists of a command based dialogue editor called MIKE which provides quick easily learned prototyping of dialogues along with refinement of the dialogue via a profile editor. The interaction is viewed as an editing of a file in STUF format which has the

expressive power of both relational and linked data structures. Display tasks are handled by formats which describe how data should appear along with editing templates which provide generic data editing and screen update procedures for various data display techniques.

The main difference between this approach and other UIMS approaches is that the interaction is viewed as a data editing process rather than simply as an input dialogue parsing problem or a screen management problem. Having adopted this view one can then identify generic classes of display and editing techniques for various data organizations. Given such a class a template is developed which carefully links the input dialogue with the display management functions to provide vastly improved interactive response. Such a class can then be applied to specific data editing problems simply by binding the parameters.

Assuming that one has an application whose data organization can suitably use the editing templates provided (which is an openended set) then an interactive browser / editor can be created in a matter of days rather than months. For those parts of such an application which do not match one of the existing templates new code can be written and easily integrated with the templates. In fact after such code has been written it should be examined for its potential to become a new template itself.

References

- [BUX83] Buxton, W. "Towards a Comprehensive User Interface Management System." *Computer Graphics* 17, 3 (July 1983).
- [GEH84] Gehani, N. *Ada: An Advanced Introduction*. Prentice-Hall, 1984.
- [GOL83] Goldberg, A. and Robson, D. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.
- [GRE85] Green, M. "The University of Alberta User Interface Management System." *Computer Graphics* 19, 3 (July 1985).
- [JAC83] Jacob, R.J.K. *Using Formal Specifications in the Design of a Human-Computer Interface*. Communications of the ACM 26, 4 (April 1983).
- [KAM83] Kamran, A. and Feldman, M.B. "Graphics Programming Independent of Interaction Techniques and Styles." *Computer Graphics* 17, 1 (Jan 1983).
- [OLS83] Olsen, D. R. and Dempsey, E. P. *SYNGRAPH: A Graphical User Interface Generator*. *Computer Graphics* 17,3 (July 83).
- [OLS85a] Olsen, D.R., Dempsey, E.P. and Rogge, R.A. "Input/Output Linkage in a User Interface Management System." *Computer Graphics* 19, 3 (July 1985).
- [OLS85b] Olsen, D.R. "Pushdown Automata for User Interface Management." *ACM Transactions on Graphics* 3, 3 (July 1985).
- [OLS85c] Olsen, D.R. "User's Manual for MIKE-2.0." Tech. Report 85-1, Computer Science Department, Brigham Young University, Provo, UT.
- [PFA85] Pfaff, G. and ten Hagan, P.J.W. *Seeheim Workshop on User Interface Management Systems*. Springer-Verlag, Berlin, 1985.

[THO83] Thomas, J. J. and Hamlin, G. Graphical Input Interaction Technique Workshop Summary. Computer Graphics 17,1 (Jan. 1983).

[VAN83] van den Bos, J., Plasmeijer, M.J. and Hartel, P.H. "Input-Output Tools: A Language Facility for Interactive and Real-Time Systems." IEEE Transactions on Software Engineering SE-9, 3 (May 1983).

AUTOMATIC GENERATION OF GRAPHICAL USER INTERFACES

Gurminder Singh and Mark Green

Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

ABSTRACT

The research reported here is focussed on the issues involved in automatically generating the presentation component of user interfaces. The design and implementation of the presentation component of the University of Alberta User Interface Management System are described. The system is used for automatically generating graphical user interfaces for interactive applications. The system has been designed to keep the other components of the user interface device independent, keep the designer's interest alive in the design process, make the design process less cumbersome, and reduce the burden of programming as far as possible. The results presented in this report are based on the experience gained through implementing a system to generate the presentation component of user interfaces automatically. The presentation component can be viewed as the lexical level of the user interface.

1. Introduction

There has been a growing awareness in software design of the importance of the user. This concern has manifested itself, for example, in analysis of desirable properties of user interfaces [Cheriton76] and in investigations into the user-friendliness of interactive systems. The concept that the user interface can be treated as a separate module within the whole system, and not simply embedded at a range of points through it, is gaining acceptability [Edmonds81]. The effort now is to make user interfaces more interactive, graphic, forgiving, and self-explanatory. But, unfortunately, the construction of good user interfaces even today remains an expensive, time-consuming, and often a frustrating process [Buxton83]. This prompted researchers in human factors to explore the possibility of automatically generating user interfaces and the notion of a User Interface Management System (UIMS). This paper describes a tool for automatically generating graphical user interfaces for interactive programs and explores the issues related to the process.

1.1. What is A User Interface?

The user interface is the part of a system that handles the interaction between the user and the other components of the system. In order to complete a useful task the system accepts inputs and presents outputs through the user interface. As more interactive systems of comparable functionality become available, their success in the market place is based increasingly on ease of use. Bad user interfaces often cause unnecessary loss of productivity and aggravation. Ease of use, not ease of implementation, has become the crucial design consideration.

The basic structure of a user interface does not change radically over a wide range of applications [Green84a]. There are a number of functions that must be performed by most user interfaces. These functions include error detection and recovery, user protocoling, and undo processing. The concepts of a separate user interface module, separate interface designer, and the common features of the user interfaces have lead to the notion of UIMS.

1.2. Automatic Generation of User Interfaces

The fact that the basic structure of a user interface does not change radically over a wide range of programs and that functions like error detection, error recovery, and help are common to almost all user interfaces leads to the idea of automatic generation of user interfaces. The high cost and large turnaround time for hand coded user interfaces provides additional motivation for the idea.

The automatic generation of the user interfaces has the following advantages:

- 1) It reduces the cost of producing user interfaces.
- 2) It provides a much shorter lead time than the hand coding of the interfaces.
- 3) The low cost and short lead time for the production of the user interfaces makes it possible to experiment with new ideas in user interface design.
- 4) Once the user interface generator is debugged completely, the software it generates is more reliable than hand coded software.
- 5) A particular user interface generator may be used to generate a number of user interfaces which are consistent in their over all approach to functions such as error reporting and help. Familiarity with one such user interface can expedite the learning of the others.

1.3. What is a UIMS?

A UIMS is a collection of software tools supporting the design, specification, implementation, and evaluation of user interfaces [Seattle83]. It performs an important role of mediating the interaction between a user and an application; satisfying user requests for application actions, and application requests for data from the user. It thus provides for the application programmer's problem specific skills to be concentrated on the application, and freed from detailed concern with managing the flow of user actions and responses. UIMSs have also been called "Dialogue Management Systems" [Roach82] or "Abstract Interaction Handlers" [Feldman82]. Over the past few years many models of UIMSs have been proposed and implemented [Newman68], [Kasik82], [Guest82], [Buxton83], [Jacob83], [Olsen Jr.83].

2. The University of Alberta UIMS

The University of Alberta UIMS [Green85], [Singh85], [Lau85], [Chia85] is based on the Seeheim model of user interfaces discussed in section 2.1. The design and implementation details of the presentation component of the U of A UIMS are described in this paper. Three main notations have been used for specifying the dialogue between the user and computer. These notations are: recursive transition networks, BNF grammars, and events. A system accepting dialogues specified by recursive transition networks is discussed in [Lau85]. Details about an event language and its implementation can be found in [Chia85]. At the present time the implementation of a grammar based notation has not been started. Support for the application interface model is currently under development.

2.1. The Seeheim Model of User Interfaces

In the Seeheim model of user interfaces [Green84b] a user interface is divided into three components as shown in Figure 1. The presentation component can be viewed as the lexical level of the user interface. This component is responsible for managing the input and output devices used by the user interface. All the interaction techniques and display formats are defined in this component. It reads the physical input devices and converts the raw input data into the form required by the other components in the user interface. The user interface employs an abstract representation for the input and output data. This representation consists of a type or name that identifies the kind of data, and the collection of values that define the data item. This chunk of information is called a token.

While the presentation component is responsible for converting user actions to input tokens, the dialogue control component defines the set of legal input tokens. It interprets the sequence of input tokens produced by the presentation component to determine the operations the user wants to perform. Once a complete command has been formed from the input tokens the dialogue control component uses the application interface model to invoke the appropriate routines in the application. Similarly the output tokens sent by the application interface model are interpreted by dialogue control and transformed into a format acceptable to the user. This component contains the control logic of the user interface. Most existing UIMSs have concentrated on this component of the user interface.

The application interface model is a representation of the functionality of the application. It represents the user interface's view of the application. The application interface model contains the descriptions of the major data structures maintained by the application, and the application routines that can be invoked by the user interface. It also covers the mode of communication between the user interface and the application. The user interface may communicate with the application in one of the three possible modes of interaction: the user initiated, the system initiated, and mixed initiative.

3. Design of the Presentation Component

The basic job of the presentation component is to convert user interactions with the input devices into input tokens, and convert output tokens into images on the output devices. The basic structure of the presentation component of the University of Alberta UIMS is presented in Figure 2. In the following sections a description of the important concepts related to the presentation component of the University of Alberta UIMS is presented.

3.1. Input Tokens

The input tokens convey information about the user's interactions with the user interface to the other parts of the UIMS. The raw data generated by the user interactions with the input devices is manipulated and restructured by the interaction techniques and control module. This new structure, called an input token, is sent to the other parts of the UIMS. An input token represents exactly one unit of information as far as the UIMS is concerned. An input token contains the following information.

- Token Number
- Token Value

The token number is the unique number assigned to each type of input token by the presentation component of the UIMS. The interpretation of the token value depends upon the token number.

3.2. Output Tokens

The output tokens are used for generating images. These tokens are generated by the dialogue control component as well as the application, and are sent to the control module of the presentation component for further processing. The control module invokes the display procedure associated with the output token and ensures that the image is generated in the appropriate window. An output token contains the following fields.

- Token number
- Token value

The token number is the unique number assigned to each type of output token by the presentation component of the UIMS. Using the token number as the key the control module finds the associated display procedure and the window name. The interpretation of the token value is left to the display procedure. Usually this field points to a structure defined in the token definition file. The token definition file contains the definitions of the structures the value field of an input or output token could point to.

3.3. Control Module

The control module is responsible for all communication with the other parts of the user interface. This communication includes sending the input tokens to the dialogue control component and receiving the output tokens from the other parts of the user interface.

The other important function of the control module is to perform an external-internal mapping. This mapping determines how the user's actions are converted into input tokens and how output tokens are converted into images. The external-internal mapping can be viewed as a dictionary used by the control module for interpreting user actions and output tokens. For input tokens the control module uses the event number and the window name of the input event to determine the input token number. In the case of output tokens, it determines the window where the image will be generated and the display procedure to be used from the output token number.

3.4. Library of Interaction Techniques

Most interaction tasks are performed by a set of interaction techniques. An interaction technique is defined as a way of using a physical input device to enter a certain type of word (command, value, location, etc.), coupled with the simplest form of feedback from the system to the user [Foley81].

There are a large number of possible interaction techniques. Each interaction technique is suitable for a particular function. The set of interaction techniques available to a designer remains very limited if he/she has to develop one every time it is required. To make the designer's choice wider a library of interaction techniques can be created. This library can be used by any designer while deciding on which interaction techniques to use. Every time a new technique is developed it can be added to the already existing library. In this way one can keep building the library and help reduce the cost and time of producing good user interfaces.

3.5. Library of Display Procedures

A display procedure is a procedure that consumes output tokens. In the process of consuming output tokens the display procedure produces images on the graphics display. This image represents the data received in the output token.

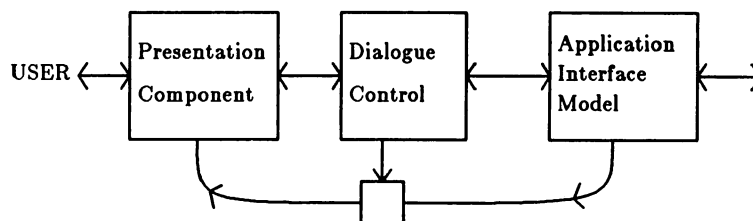


Fig. 1. The Seeheim Model of a User Interface

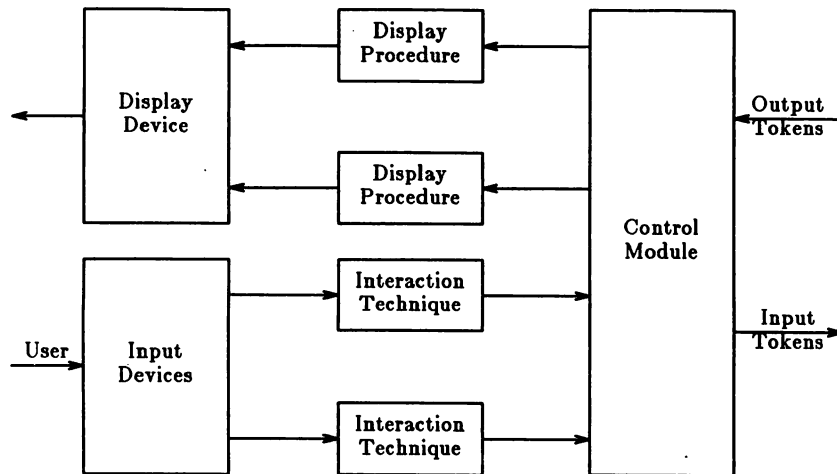


Fig. 2. The Structure of the Presentation Component

Each display procedure has a specific purpose and is used to generate a specific image. Examples of display procedures are Angle Display, Vertical Bar Display, and Text Windows.

The idea of having a library of display procedures is similar to that of the library of interaction techniques. By adding new display procedures to the library a large body of procedures can be built. This library can then be used for fast and economical production of user interfaces.

4. The Implementation

The presentation component of the University of Alberta UIMS has been implemented on VAX 11/780 running UNIX* 4.2 BSD. The programs used for implementing the system are written in the programming language C.

4.1. Structure of the Presentation Component

The presentation component of the University of Alberta UIMS is responsible for the following activities.

- Screen management
- Information display
- Associating interaction techniques with windows
- Associating display procedures with output tokens
- Assigning unique token numbers to input and output tokens
- Converting user interactions into input tokens
- Converting output tokens into images
- Lexical feedback
- Adapting to different display devices, if possible

An interactive approach to the design of the presentation component of the user interfaces has been adopted in this UIMS. There are two steps involved in the complete design. The first step is the specification step. In this step the user interface designer interactively specifies the design information. This information is then used in the second step for generating the presentation component of the user interface and providing run-time support. To support both these functions the presentation component of the University of Alberta UIMS is divided into two logically independent parts. The first part, called "ipcs" (interactive presentation component specification), accepts the design specifications from the designer and generates a data base, token tables, and 'C' procedures. The second part, called "pcg" (presentation component generation), consists of a number of procedures which provide run-time support for the presentation component of the user interfaces. This part is driven by the data base and the token tables. The 'C' procedures produced by ipcs are compiled and linked with the pcg procedures. The entire sequence of creating a presentation component is

*UNIX is a trademark of AT&T Bell Laboratories.

shown in Figure 3. In implementing the presentation component a graphics package called WINDLIB [Green84c] and a data base package called FDB [Green83] are extensively used.

4.2. The Specification Step

The complete specification of the presentation component involves the specification of the following components.

- Screen layout
- Input tokens and interaction techniques associated with windows
- Output tokens and display procedures
- Menu layouts

The interactive specification program "ipcs" is used to enter the design information. The ipcs screen is divided into four areas as shown in Figure 4. The work area corresponds to the display screen of the user interface being designed. The designer positions windows and menus in this area. Above the work area is a text area used for help and error messages. The right side of the screen is used for the ipcs menu. An area across the bottom of the screen displays some of the attributes of the current window in the work area. In the following sections a brief description of the functions performed by ipcs is presented.

4.2.1. Window Definition

ipcs starts off by displaying the layout shown in Figure 4. At the start of the specification session the work area corresponds to the device window on which the user interface will be implemented. All the windows created by the user interface designer are children of this window. To start defining windows the designer selects the "Window Definition" command from the ipcs menu. A window can then be defined by pointing at its two opposing corners. Once a window is defined it can be removed, stretched to a different size, or moved to a new position. An arbitrary number of windows can be defined at each level.

4.2.2. Window Attributes

The window attributes can be defined by selecting the "Window Attributes" command from the ipcs menu. This command assigns and displays default attributes for each window in the work area. The default attributes consist of the window name, window limits, background colour, drawing colour, and boundary colour. The value of an attribute can be changed by pointing at it and entering a new value.

In this step an interaction technique can be associated with a window. An interaction technique can be selected from the library of interaction techniques, or it can be a procedure written by the user interface designer. The name of the output token associated with the window is also specified in this step. On receiving this token the run-time support

module creates the window. It is important to note that it is the output token name, not the window name, which is used by the run-time support module.

ipcs is capable of handling windows of variable size. In a normal case, depending upon the size of the window, one, five, or ten window attributes are displayed at a time. In the case of one or five attributes, the next set of attributes can be displayed by moving the tracking cross inside the window and hitting carriage return. To be able to define or change attributes for a very small window, its size can be temporarily adjusted. The size of such a window is adjusted for the purposes of display only.

The system also allows the use of overlapping windows. In the case of overlapping windows, the attributes displayed in one window overlap with the attributes in the other window. These windows can be flipped by pointing at the desired window. The selected window temporarily becomes the top-most window and its attributes become visible.

It is not necessary to complete the definition of all the attributes of a window at one time. The designer can postpone the definition of all or some of the window attributes for a later time. The system does not force any pre-determined sequence of specification steps to be followed.

The attributes for a window can be changed as often as desired. The system does not differentiate between changing a default attribute, or a designer-defined attribute. This facilitates the interactive design of user interfaces. This mechanism for accommodating changes in the specification also helps in adapting user interfaces to individual users. The interaction technique or colours associated with a window, for example, can be easily changed to the actual user's liking.

4.2.3. Menu Definition

The "Menu Definition" command is used for defining menus. A menu is always associated with one of the windows defined in the work area. A menu consists of a menu header and a variable number of menu items. The menu header contains information affecting the appearance and location of the menu. This information consists of menu name, menu type, menu items placement option, menu orientation, and menu output token. Menu name is entered by the designer. A meaningful menu name can be used to remember the purpose or contents of the menu. The system provides facilities for fixed as well as pop-up menus. The default menu type is "fixed". The placement of menu items can be automatically handled by the system at the run time. The menu area is equally divided and allocated to each menu item in the menu. The system then centers the text and icons. The designer, however, has the option of specifying the location and size of individual menu items in the menu. The default menu items placement option is "system", it can be changed to "designer". The default menu orientation is "vertical", it can be changed to horizontal. Each menu is assigned an output

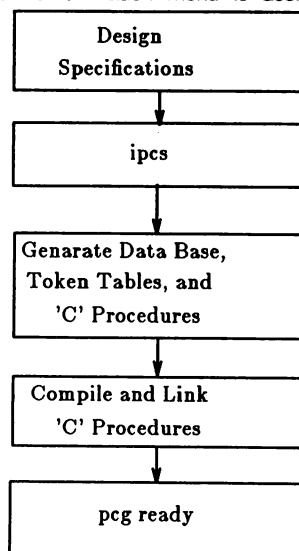


Fig. 3. Sequence for Constructing a Presentation Component

token. On receiving this output token the run-time support module displays the menu. The run-time support module recognizes the menus by their output token, not by the menu names.

A variable number of menu items can be associated with a menu. Each menu item occupies a rectangular area within the menu area. A menu item can be labeled either by an icon or text strings. A menu may consist of a mixture of iconic and textual menu items. For each menu item the designer specifies its type; iconic or textual. In the case of textual menu items one or more lines of text can be associated with the item.

In the case of icons the system provides a library of icons. An icon may be selected from this library, or it can be produced by using an interactive editor, called ICON [Green86], developed at the University of Alberta. To associate an icon with a menu item the name of the procedure drawing the icon is entered.

The system allows the designer to associate more than one menu with a window. This helps in creating a hierarchy of menus. It is important to note that the menus may be displayed in any order. It is not necessary to display the menus in the order they are defined. Therefore, though the menu definition hierarchy is simple, the menu display hierarchy can be as complex as desired.

4.2.4. Input Token Definition

Input tokens can be associated with a window by selecting the "Input Token Definition" command from the ipcs menu. The input tokens convey information about the user's interactions with the user interface to the other parts of the UIMS. A variable number of input tokens can be associated with a window. An input token definition consists of token name and the associated event number. The system allows the designer to delete or add any number of input tokens during a specification session.

4.2.5. Output Token Definition

Output tokens can be associated with a window by selecting the "Output Token Definition" command from the ipcs menu. For output tokens the designer specifies the name of the token along with the name of a display procedure. On receiving the output token, the run-time support module invokes the associated display procedure. Based on the information contained in the output token, the display procedure produces the image in the window in which the output token is specified.

More than one output token can be associated with a window. Different output tokens are used to produce different images in the same window. The system allows the designer to modify or delete an output token after its definition.

4.2.6. Next and Previous Level Definitions

To be able to define the next level in the tree of windows, the designer can select "Next Level" command from the ipcs menu and point to a window in the work area. This window now becomes the new parent. Some of the important attributes of this window are displayed in the bottom of the screen and the environment switches back to the one shown in Figure 4.

A tree of windows can be created by using the "Next Level", "Previous Level", and "Window Definition" commands from the ipcs menu. By selecting the "Next Level" command and pointing at a window in the work area, children of the window pointed at can be created. The work area corresponds to the selected window and child windows can be created using the "Window Definition" command.

The system does not require the designer to complete the definition of the current level before going to the next level in the tree of windows. The tree can be defined branch-by-branch, level-by-level or by a mixture of the two approaches. This flexibility in designing the tree allows the designer to work more methodically and concentrate on one part of the user interface at a time. The system does not put any limit on the depth of the tree or number of windows at a particular level of the tree.

4.3. Novice and Expert Users of ipcs

ipcs has two levels of use; "novice" and "expert". A novice may use only the basic set of commands. In this mode

To change an attribute, move the cursor on top of the attribute and type the new value. Hit carriage return after entering the value.
 FF2 - to adjust size, FF3 - to flip windows, FF4 - to redraw

Name: <input type="text" value="w9"/> Limit ltr: <input type="text" value="0.00"/> Limit ltr: <input type="text" value="0.00"/> Limit urx: <input type="text" value="1.00"/> Limit ury: <input type="text" value="1.00"/> Bg Color: <input type="text" value="0"/> Draw Color: <input type="text" value="1"/> Endry Color: <input type="text" value="1"/> Intret Tech.: <input type="text" value="none"/> Output Token: <input type="text" value="w9"/> <hr/> Name: <input type="text" value="w7"/>	Name: <input type="text" value="w8"/> Limit ltr: <input type="text" value="0.00"/> Limit ltr: <input type="text" value="0.00"/> Limit ltr: <input type="text" value="0.00"/> Limit urx: <input type="text" value="1.00"/> Limit ury: <input type="text" value="1.00"/> Bg Color: <input type="text" value="0"/> Draw Color: <input type="text" value="1"/> Endry Color: <input type="text" value="1"/> Intret Tech.: <input type="text" value="none"/> Output Token: <input type="text" value="w8"/>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Window	Parent	Bg Color	Draw Color	Boundary	Int. Tech.	In Tokens	Out Tokens
jup7	jup7	0	1	1	none	0	0

Window

Definition

Window

Attributes

Menu

Definition

Input Token

Definition

Output Token

Definition

Next

Level

Previous

Level

EXIT

Help

ON OFF

Fig. 4. The ipcs Screen Layout

all the input is done through one button on the cursor puck, and all through the ipcs session the designer is guided by help messages. The help messages are quite detailed for the novice. In expert mode, the designer is allowed to use other buttons on the puck, and the messages produced by the system are terse. The use of extra buttons on the puck allows the designer to delete, resize, move, or temporarily change the size or priority of a window. A profile for each user of ipcs (called "userprofile") is created and maintained by the program, initially tagging each user as a novice. The userprofile file stores the status ("novice" or "expert") of the user, the number of times he has invoked ipcs, and the number of times the designer successfully finished the specification sessions. A novice is upgraded to an expert based on the number of successful executions of the ipcs.

4.4. Output of ipcs

The output produced by ipcs is used to drive the run-time support module, and provides the interface for the dialogue control component and the application interface model. The information produced for the run-time support module consists of an FDB data base and 'C' procedures, whereas the interface information for the dialogue control component and application interface model consists of tables of input and output tokens. A brief description of these files is presented in the following sub-sections.

4.4.1. Data Base Description

The design information for the presentation component is stored in an FDB data base. An object in the data base is represented by a frame. A window frame points to the next window, at the same level as well as to its child windows at the next level. In addition, a window frame points to the first menu, first input token, and first output token frames associated with the window. Each of these in turn point to the frames providing detailed information about the objects.

4.4.2. 'C' Procedures

The 'C' procedures generated by ipcs are mainly used for passing parameters to the interaction techniques. The procedure calls are also used for loading the appropriate routines from the library of interaction techniques, display procedures, and icons. These procedures are compiled and linked with the other routines for run time support.

4.4.3. File of Input/Output Tokens

This file contains the names of all the input and output tokens along with a number. These numbers are assigned by ipcs to each of the input and output tokens. For reasons of efficiency this number is used for communication amongst various components of the user interface at run time.

4.5. Run-Time Support Module

The run-time support for the presentation component of the user interface is provided by the routines in "pcg". The following functions are supported by pcg.

- Receive the user's interactions in the form of WINDLIB events, reformat them into input tokens, and send these tokens for further processing.
- Receive output tokens from the dialogue control component, find the appropriate window and display procedure, and call the display procedure.
- Display menus and highlight the selected menu items.

The program pcg is driven by the data base created by the specification program ipcs. It retrieves the information from the database and restructures the information as required. It also receives some help from the 'C' code generated by the ipcs. This code is compiled and linked with the other run time routines. Pcg is divided into three logical parts. The first part is responsible for displaying menus and performing the associated bookkeeping. The second part handles the user interactions and generates the input tokens. The third part receives the output tokens and is responsible for their display.

5. Conclusions

In our system attention is focussed on the issues involved in the automatic generation of presentation components of user interfaces. The separation of presentation component from the dialogue control component helps designers work more methodically, and may therefore result in better user interfaces. The approach also overcomes one of the major stumbling blocks in user interface design, namely, the representation of geometric information in textual form. In our design most of the geometrical information is entered graphically. The system provides a window based environment which helps designers structure the user interfaces in a more natural way. The system also provides facilities for creating and maintaining a hierarchy of windows and menus. The interactive design of user interfaces is supported by allowing the designer to move to the next level in the hierarchy without completing the definition of all aspects of the user interface at the current level. The system provides more freedom to the designer by not imposing any predetermined sequence of commands for creating user interfaces.

The second contribution of this system is to show that all device dependencies can be limited to the presentation component of the user interface. If the user interface is moved to a different device only the presentation component needs to be changed. This increases the portability of the user interfaces. Also, the presentation component can be designed to support a range of devices, and automatically adapt to the one in use without changing the structure of the dialogue.

The third contribution of this system is to show that user interfaces can easily be adapted to individual users. Screen layout, for example, can be easily tailored for left handed users. The selection of interaction techniques and display formats can also be easily changed to the actual user's liking.

It is also observed that the existence of a separate presentation component encourages the use of a standard library of interaction techniques. This speeds up the process of generating user interfaces to a great extent and reduces the cost of programming considerably. This reduction in cost and time encourages experimentation with user interfaces and hence increases user satisfaction.

References

- Buxton83.
W. Buxton, M. R. Lamb, D. Sherman and K. C. Smith, Towards a Comprehensive User Interface Management System, *Computer Graphics* 17, 3 (July 1983), 35-42.
- Cheriton76.
D. R. Cheriton, Man-Machine Interface Design for Time-Sharing Systems, *Proc. of ACM Annual Conf.*, 1976, 362-366.
- Chia85.
M. S. Chia, *An Event Based Dialogue Specification for Automatic Generation of User Interfaces*, M.Sc. Thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, Canada, 1985.
- Edmonds81.
E. A. Edmonds, Adaptive Man-Computer Interfaces, in *Computing Skills and the User Interfaces*, M. J. Coombs and J. L. Alty (ed.), Academic Press, London, 1981, 389-426.
- Feldman82.
M. Feldman and G. Rogers, Towards the Design and Development of Style Independent Interactive Systems, *Proc. 1st Annual Conf. on Human Factors in Computer Systems, Gaithersburg Maryland*, Mar. 1982, 111-116.
- Foley81.
J. D. Foley, V. L. Wallace and P. Chan, *The Human Factors of Interaction Techniques*, IIST Report 81-03, Dept. of Electrical Engineering and Computer Science, The George Washington University, Washington, D.C., Mar. 1981.
- Green83.
M. Green, M. Burnell, H. Vernjak and M. Vernjak, Experience with a Graphical Data Base System, *Proc. Graphics Interface '83*, 1983, 257-270.
- Green84a.
M. Green, *The Design of Graphical User Interfaces*, Ph.D. Thesis, Univ. of Toronto, Toronto, Canada, 1984.
- Green84b.
M. Green, Report on Dialogue Specification Tools, *Computer Graphics Forum* 3, (1984), 305-313.
- Green84c.
M. Green and N. Bridgeman, *WINDLIB Programmer's Manual*, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, Canada, 1984.
- Green85.
M. Green, The University of Alberta User Interface Management System, *Proc. Siggraph' 85*, 1985, 205-213.
- Green86.
M. Green and G. Singh, Windows as a User Interface Structuring Mechanism, (*in preparation*), 1986.
- Guest82.
Stephen P. Guest, The Use of Software Tools for Dialogue Design, *Int. Journal of Man-Machine Studies* 16, (1982), 263-285.
- Jacob83.
R. J. K. Jacob, Executable Specifications for a Human-Computer Interface, *Proc. CHI'83*, Dec. 1983, 28-34.
- Kasik82.
D. J. Kasik, A User Interface Management System, *Computer Graphics* 16, 3 (July 1982), 99-106.
- Lau85.
S. C. Lau, *The Use of Recursive Transition Networks for Dialogue in User Interfaces*, M.Sc. Thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Canada, 1985.
- Newman68.
W. M. Newman, A System for Interactive Graphical Programming, *Proc. Spring Joint Computer Conf.*, 1968, 47-54.
- Olsen Jr.83.
Dan R. Olsen Jr. and Elizabeth P. Dempsey, SYNGRAPH: A Graphical User Interface Generator, *Computer Graphics* 17, 3 (July 1983), 43-50.
- Roach82.
J. Roach, R. Hartson, R. Ehrich, T. Yuntten and D. Johnson, DMS: A Comprehensive System for Managing Human-Computer Dialogue, *Proc. 1st Annual Conf. on Human Factors in Computer Systems, Gaithersburg Maryland*, Mar. 1982, 102-105.
- Seattle83.
Seattle, Proc. of Graphics Input Interaction Technique Workshop, June 2-4 Battelle Seattle, *Computer Graphics*, Jan. 1983.
- Singh85.
G. Singh, *Presentation Component for the U of A UIMS*, M.Sc. Thesis, Dept. of Computing Science, Univ. of Alberta, Edmonton, Canada, 1985.

A FAST ALGORITHM FOR GENERAL RASTER ROTATION

Alan W. Paeth

Computer Graphics Laboratory, Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
Tel: (519) 888-4534, E-Mail: AWPaeth%watCGL@Waterloo.CSNet

ABSTRACT

The rotation of a digitized raster by an arbitrary angle is an essential function for many raster manipulation systems. We derive and implement a particularly fast algorithm which rotates (with scaling invariance) rasters arbitrarily; skewing and translation of the raster is also made possible by the implementation. This operation is conceptually simple, and is a good candidate for inclusion in digital paint or other interactive systems, where near real-time performance is required.

RÉSUMÉ

La rotation d'un "raster" d'un angle arbitraire est une fonction essentielle de plusieurs logiciels de manipulation de "raster". Nous avons implémenté un algorithme rapide de rotation de "raster" conservant l'échelle de l'image. Nous d'écrivons ce système qui permet aussi le biaisage et la translation du "raster". Cette opération, d'un concept simple, se révèle un bon candidat à l'insertion dans un logiciel de "paint system" (ou autre système interactif) où une performance quasi-temps réel est nécessaire.

Keywords: raster rotation, frame buffer, real-time.

INTRODUCTION

We derive a high-speed raster rotation algorithm based on the decomposition of a 2-D rotation matrix into the product of three shear matrices. Raster shearing is done on a scan-line basis, and is particularly efficient. A useful shearing approximation is averaging adjacent pixels, where the blending ratios remain constant for each scan-line. Taken together, our technique rotates (with anti-aliasing) rasters faster than previous methods. The general derivation of rotation also sheds light on two common techniques: small angle rotation using a two-pass algorithm, and three-pass 90-degree rotation. We also provide a comparative analysis of Catmull and Smith's method [Catm80] and a discussion of implementation strategies on frame buffer hardware.

STATEMENT OF THE PROBLEM

A general 2D counter-clockwise rotation of the point (x, y) onto (x', y') by angle θ is performed by multiplying the point vector (x, y) by the rotation matrix:

$$M = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

The matrix is orthogonal: it is symmetric, rows and columns are unit vectors, and the determinant is one. To rotate a raster image, we consider mapping the unit cell with center at location $(1, j)$ onto a new location $(1', j')$.

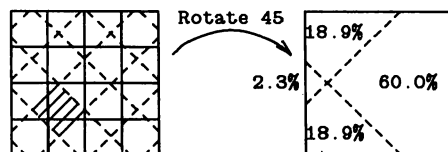


Figure 1. Rotation by Raster Sampling

The image of the input cell on the output grid is a cell with (usually) a non-integral center, and with a rotation angle theta (θ). We adopt a "box-filter" sampling criterion, so the value of the output pixel is the sum of the intensities of the covered pixels, with each contributing pixel's intensity weighted in direct proportion to its coverage (Figure 1). Note that the output pixel may take intensities from as many as six input pixels. Worse, the output pixel coverage of adjacent input pixels is non-periodic; this is directly related to the presence of irrational values in the rotation matrix. Clearly, the direct mapping of a raster by a general 2x2 matrix is computationally difficult: many intersection tests result, usually with no coherence or periodicity to speed program loops.

ROTATION THROUGH SHEARING

Now consider the simplest 2x2 matrices which may operate on a raster. These are shear matrices:

$$\text{X-shear} = \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \quad \text{Y-shear} = \begin{bmatrix} 1 & 0 \\ \beta & 1 \end{bmatrix}$$

Shear matrices closely resemble the identity matrix: both have a determinant of one. They share no other properties with orthogonal matrices. To build more general matrices, we form products of shear matrices — these correspond to a sequence of shear operations on the raster. Intuitively, consecutive shearing along the same axis produces a conforming shear. This follows directly:

$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & a' \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & a+a' \\ 0 & 1 \end{bmatrix}$$

Thus, shear products may be restricted to products of alternating x and y shears, without loss of generality. The product of three shears gives rise to a general 2x2 matrix in which three arbitrary elements may be specified. The fourth element will take on a value which insures that the determinant of the matrix remains one. This "falls out" because the determinant of the product is the product of the determinants (which are always one for each shear matrix). Orthogonal 2x2 matrices also have unit determinant, and may thus be decomposed into a product of no more than three shears.

$$\begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \beta & 1 \end{bmatrix} \begin{bmatrix} 1 & \gamma \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Solving the general equation, we have $\alpha = \gamma = 1 - \cos\theta / \sin\theta$; $\beta = \sin\theta$. The first equation is numerically unstable near zero, but can be replaced by

the half-angle identity: $\alpha = \gamma = -\tan(\theta/2)$. Program code to shear and update the point (x,y) with (x',y') is then:

```
/* X Shear */      /* Y Shear */
x' := x - sinθ * y; x' := x;
y' := y;           y' := y + tan(θ/2) * x;
```

When the output vector replaces the input, $x \equiv x'$ and $y \equiv y'$, so the second line of the sequence may be optimized out. Consecutive shears yield sequential program steps. Thus, a three-shear rotation is achieved by the three program statements:

$$x := x + \alpha * y; \quad [1]$$

$$y := y + \beta * x; \quad [2]$$

$$x := x + \alpha * y; \quad [3]$$

With $\theta \approx 0$, Cohen [Newm79] uses steps [1] and [2] to generate circles by plotting points incrementally. His derivation begins by choosing α and β to approximate the conventional rotation matrix, and then points out that by reassigning $x + \alpha * y$ to the original variable x in [1], and not to a temporary value x', the determinant becomes one, and the circle eventually closes. Our analysis demonstrates formally why this is true: rewriting the variables constitutes a shear, and the sequence of shears always maintains a determinant of one. Augmenting the code with line [3] would convert the two-axis shear into a true rotation: the circle generator would then produce points rotated through a constant angle relative to the preceding point. This is important should the algorithm be used to produce circle approximations as n-gons (and not point drawings), where $\theta = 360/n$ is no longer small.

RASTER SHEARING

Raster shearing differs from point transformation because we must consider the area of the unit cell which represents each pixel. Fortunately, the shear operation modifies the pixel location with respect to only one axis, so the shear can be represented by skewing pixels along each scan-line. This simplifies the intersection testing that must go on to recompute the intensity of each new output pixel.

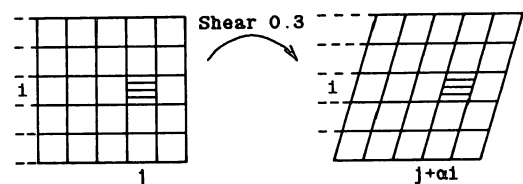


Figure 2. Raster Shearing Along the X Axis

In general, the unit square $P(1, j)$ on row 1 is rewritten as a unit parallelogram with side of slope $1/\alpha$ on row 1, with the former displaced by αy pixel widths. This displacement is not usually integral, but remains invariant for all pixels on the 1th scan-line. For illustration and implementation, it is represented as the sum of an integral and a fractional part ("f" in Figure 3; "skew" in Figure 6). Those pixels covered by this parallelogram are written with fractional intensities proportional to their coverage by the parallelogram. The sum of all these pixels must equal the value of the original input pixel, as they represent this input pixel after shearing.

We next approximate this parallelogram of unit area with a unit square. Placing the edges of the square through the midpoints of the parallelogram, we produce an exact approximation when the parallelogram covers two pixels, but not when it covers three. This approximation is the basis for our rotation algorithm. As we shall see, it can be implemented as a very efficient inner-most pixel blending loop, thus offsetting the cost of making three shearing passes, as compared to previous techniques, which employ two less efficient (though more general) passes.

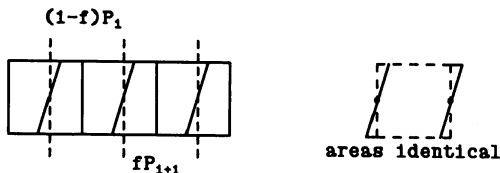


Figure 3. The Parallelogram Approximation

Based on this filtering strategy, we consider two approaches to rotation. First, we seek angles θ for which the filtering is exact. Second, we analyze the filter for arbitrary values of θ where the filter may not be exact.

Filtering is exact when all parallelograms overlap no more than two pixels. This will always occur when the shear offset is of length $1/n$, as a periodic cycle of n parallelograms results, in which each spans exactly two pixels. Choosing this ideal filter for the first and third passes, we derive the second pass shear value. Setting $\alpha=1/n$, we have $\theta=2 \cdot \tan^{-1}(1/n)$ and thus by manipulation of inverse trigonometric functions, $\beta=2n/(1+n^2)$. Setting $n=1$ yields $\alpha=-1$, $\beta=1$ and thus rotations with $\theta=90$ degrees are exact (not possible with the Catmull-Smith approach). Moreover, this shearing matrix never generates fractional values (implying no anti-aliasing must take place), so 90 degree rotation may be coded as a three pass pixel "shuffle".

Other choices of n yield exact sampling in the two α passes, as $\alpha=1/n$. Here $\beta=2n/(1+n^2)$ (in the middle pass) will never be of the form $1/m$, so some filtering artifacts will be present. However, we can form small rational values for α and β corresponding to various angular rotations, and create specialized filters, in which only the β pass generates small errors on a periodic basis. When α and β are small rationals of the form $1/j$, then the shear values (which are used as blending coefficients by our algorithm) will recur every j scan-lines, leading to customized algorithms. Solving for general rational values of α and β , we find that $\alpha=1/j$ and $\beta=2j/(1+j^2)$. These tabulated values give rise to highly efficient filters, with approximation errors minimized:

α	β	θ
-1	1	90.00
-3/4	24/25	73.74
-2/3	12/13	67.38
-1/2	4/5	53.13
-1/3	3/5	36.87
-1/4	8/17	28.07
-1/5	5/13	22.62

Figure 4. Rotation by a Rational Shear

We now consider arbitrary choices of θ , and then the precision of the rotation. For $\theta > 90$ degrees, our shear parallelogram may span four pixels, and the filtering rapidly breaks down. Based on the four-fold symmetry of a raster, we may restrict our attention to rotations of no more than 45 degrees, where our approximation has worst-case performance (because α and β grow monotonically with $0 \leq \theta < 90$ degrees). Here $\alpha=1-\sqrt{2} \approx -.4142$; and $\beta=\sqrt{2}/2 \approx .7071$. The second β pass is the most error-prone.

Probabilistically, its filter is exact 29.3% of the time. Otherwise, the parallelogram spans three pixels, and the error, as a function of fractional pixel skew, grows quadratically to a cusp, reaching its worst-case error when the parallelogram is symmetric about the output pixel. This error is $\sqrt{2}/8$ or 17.7%. However, the sampling tile shifts as the shear value changes with each scan-line, so average degradation to the sheared raster is computed by integrating over parallelograms of all possible skew. Solving the equations, we find that the worst-case shear filter approximates intensities to within 4.2% of the actual intensity. For rotations less than 45 degrees, the approximation is even closer, as the probability of the parallelogram spanning three pixels decreases. Where it does, the error terms are also smaller.

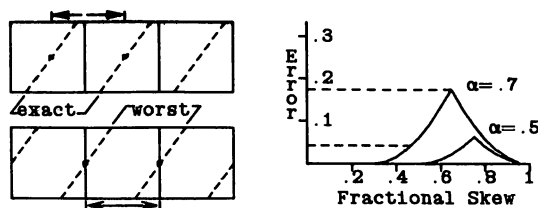


Figure 5. Approximation Error

The nature of the error is to concentrate intensities from a center pixel whereas the true box-filter approximation calls for contributing coverages from two neighboring pixels. Thus, the approach "peaks" the data: the nature of the degradation is not random. Further, a reasonable implementation of the filter guarantees that when any scan-line is skew-sheared by a fractional amount, the contributing intensities of each input pixel sum to 1.0 — the filter parallelograms never overlap. If we consider the sum of the pixel intensities along any scan-line, this sum remains unchanged after the shear operation. Thus, the algorithm produces no visible shifts in intensity, and introduces no "holes" during rotation. The only rotation artifacts discernible appear with high-frequency data (such as lines of single pixel width), and even then only after magnification. This property is shared generally with rotation and translation algorithms which must resample such "sharp" rasters onto non-integral pixel boundaries.

IMPLEMENTATION

Scan line shearing is approximated by a blending of adjacent pixels. In the following code segment, the "pixmult" function returns a pixel scaled by a value skewf, where $0 \leq \text{skewf} < 1$, is a constant parameter for all "width" passes through the inner-most loop:

```
PROCEDURE xshear(shear, width, height)
  FOR y := 0 TO height-1 DO
    skew := shear * (y+0.5);
    skewl := floor(skew);
    skewf := frac(skew);
    oleft := 0;
    FOR x := 0 TO width-1 DO
      pixel := P(width-x, y);
      left := pixmult(pixel, skewf);
      /* pixel - left = right */
      pixel := pixel - left + oleft;
      P(width-x+skewl, y) := pixel;
      oleft := left;
    OD
    P(skewl, y) := oleft;
  OD
```

Figure 6. Shearing Algorithm for X Axis

This operation shears a raster of size (width, height) by the value present in "shear", so the data matrix P must be of sufficient width to accommodate the shifted output data. Note that only "width" output entries are written, so the skewed output line may be written to frame buffer memory modulo the frame buffer pixel width, thus requiring no additional memory, but complicating the specification of data to the three shear passes. A virtual frame buffer implementation which provided a notion of "margins" to active picture detail can maintain this offset information implicitly.

A shear operation always has an axis of shear invariance (it is an fact an eigenvector). In this implementation, the axis is the pixel boundary "below" the final row of pixel data at a distance "height". This gives rise to rotation about the interstices between pixel centers. To rotate rasters about pixel centers, the "0.5" half-pixel offset may be removed.

The code splits each pixel into a "left" and "right" value using one multiply per pixel; left and right always sum exactly to the original pixel value, regardless of machine rounding considerations. The output pixel is then the sum of the remainder of the left-hand pixel, plus the computed fractional value for the present (right-hand) pixel. The "pixmult" function reduces to a fractional multiply or table lookup operation with monochromatic images. More generally, it may operate on an aggregate pixel which might contain three color components, or an optional coverage factor [Port84]. Because read and write references to P occur at adjacent pixel locations during the course of the inner-most loop, pixel indexing can be greatly optimized.

On machines lacking hardware multiply, code to shear a large (512x512) image may build a multiply table at the beginning of each scan-line, and then use table lookup to multiply. By skew symmetry, x-shearing of line -n and line n are identical, save for shear direction, so one table may be used for two scan-lines, or for every 1024 pixels. With a pixel consisting of three 8-bit components, the table length is 256, and table fetches will exceed table loads by a factor of 12. Since the table can be built with one addition per (consecutive) entry, its amortized cost per lookup is low, and decreases linearly with raster size.

Framebuffers are beginning to incorporate integer multiply hardware, often targeted to pixel blending applications (The Adage/Ikonas frame buffers at Waterloo's Computer Graphics Laboratory provide a 16-bit integer multiply in hardware). This speeds the evaluation of the pixel blending; the majority of the inner-loop overhead is in (un)packing the 24-bit RGB pixel to provide suitable input for the multiplier. Fortunately, the addition used to complete the blend may be done as a 24-bit parallel add, because the values to be summed, "left" and "right", have been scaled by frac and 1-frac respectively. Thus, the

blending operation is "closed", and no carry can overflow from one pixel component into the next.

Finally, the shear code may more generally be used to introduce spatial translation of the raster. By introducing an output offset in the shear code, a "BitBlt" [Inga78] style operation may be included at no extra cost. In this setting, "skewi" and "skewf" would have integral and fractional offsets added to them to accommodate the lateral displacement of the raster. Displacement during data passes two and three provides arbitrary displacement on the plane, with orthogonal specification of the displacement parameters.

More generally, when the code is incorporated into a larger package which provides arbitrary (affine) matrix operations on a raster, the composite of all intermediate image transformations are represented in one matrix. This avoids unnecessary operations to the image. Eventually, this matrix is decomposed into three operations: scaling, rotation and shearing (plus an optional translation if a 3x3 homogeneous matrix is used). The shearing, rotation and possible translation operations may be gathered into one three-shear operation. The scale pass prefaces the rotation if it scales to a size larger than 1:1, otherwise it follows the rotation. This maximizes image quality and minimizes data to the shear (and possibly rotate) routines. Other four pass scale/shear sequences are discussed in the literature [Weim80].

COMPARISONS

As with the Catmull-Smith approach, the algorithm may be implemented as a pipeline for real-time video transformation. Both approaches require two "rotators" to transpose the data entering and leaving the second scan-line operator, as this step requires data in column (and not row) order.

In that approach, two scan-line passes (by x, then by y) are made upon the input raster. These may be modeled by the matrix transformation:

$$\begin{bmatrix} 1 & 0 \\ \tan\theta & \cos 2\theta \sec\theta \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

These slightly more general matrices perform a simultaneous shear and scale along one axis, while leaving the second axis unchanged. This approach saves one data pass, but incurs the penalty of more complex scan-line sampling.

Because sample pixels are both sheared and scaled, no pixel-to-pixel coherence of fractional sampling location exists. Thus, each pixel must be sampled at two fractional locations, doubling the number of pixel (aggregate RGB) multiplies for each pass. Hand analysis of our microcode showed that this is already the dominant operation in the pixel loop.

Further, the Catmull-Smith approach must additionally recompute the fractional sample points for each next pixel, or approximate their location using fixed-point arithmetic. In our implementation, fractional sampling points are constant per scan-line, and are calculated exactly in floating point at the beginning of each line.

Compared generally to other work, our algorithm finds application where a generalized "BitBlt" operation is needed to perform rotation and translation efficiently. More complex pixel sampling passes may justify their added expense in allowing for generalize rotation operations, such as Krieger's modified two-pass approach [Krie84] used to perform 3-D rotation with perspective transformation, useful in texture mapping.

CONCLUSIONS

The technique outlined here performs arbitrary high-speed raster rotation with anti-aliasing and optional translation. The mathematical derivation guarantees scaling invariance when rotating. The implementation strategy allows for particularly fast operation, while minimizing the approximation error. This algorithm is a powerful tool in the repertoire of digital paint and raster manipulation systems. Coupled with state-of-the-art raster scaling techniques, it can transform an input raster by an arbitrary 2x2 transformation matrix in near real time.

REFERENCES

- [Catm80] Catmull, E., Smith A. R. "3-D Transformations of Images in Scanline Order" *ACM Computer Graphics* (SIGGRAPH '80) 14(3), July 1980, pp. 279-285.
- [Newm79] Newman, W.M., Sproull, R.F. *Principles of Interactive Computer Graphics* (2nd ed), pp. 28 McGraw-Hill, New York 1979.
- [Inga78] Ingalls, D.H.H. "The Smalltalk-76 programming system: design and implementation" *Fifth ACM Symp. Prin. Prog. Lang.*, January, 1978, pp. 9-16.
- [Krie84] Krieger, R.A. "3-D Environments For 2-D Animation", MMath Essay, University of Waterloo, Ontario, 1984.
- [Port84] Porter, T., Duff, T. "Compositing Digital Images" *ACM Computer Graphics* (SIGGRAPH '84) 18(3), July, 1984, pp. 253-259.
- [Weim80] Weiman, C.F.R. "Continuous Anti-Aliased Rotation and Zoom of Raster Images" *ACM Computer Graphics* (SIGGRAPH '80) 14(3), July 1980, pp. 291.

A CEL-BASED MODEL FOR PAINT SYSTEMS

Terry M. Higgins and Kellogg S. Booth

Computer Graphics Laboratory, Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
Tel: (519) 888-4534, E-Mail: KSBooth%watCGL@Waterloo.CSNet

ABSTRACT

We present a comprehensive formal model for a computer paint system providing capabilities beyond those of traditional designs. The system incorporates an *alpha* channel to enable artwork to have variable opacity in a manner reminiscent of "cel painting." Operations which may be performed on these RGBA images include digital painting, airbrushing, erasing, masking, and image compositing. These are implemented as instances of the digital compositing algebra introduced by Duff and Porter. Our implementation model extends a proposal by Tanner, *et al.* It is cost-effective and is based on the concept of a virtual frame buffer containing a higher-level description of the image being painted and an associated output transformation that maps the contents into a standard RGB frame buffer used only for viewing. Ways of implementing the model to take advantage of multiprocessing capabilities in various host and frame buffer architectures are discussed and three implementations are examined.

Keywords: *brush, cel, digital compositing, mask, multiprocessor, output transformation, paint, virtual frame buffer, RGBA.*

RÉSUMÉ

Nous présentons un modèle formel décrivant un logiciel de palette de couleurs électronique ("paint system" offrant des possibilités allant au delà des modèles traditionnels. Ce système comprend un canal *alpha* qui permet au dessin d'avoir une opacité variable ressemblant à la technique d'animation appelée gouachage de cellos. Les opérations pouvant être exécutées sur ces images RGBA comprennent: dessin digital, airbrushing, effaçage, masquage et composition d'images; ces opérations sont implémentées suivant l'algèbre digital de composition d'écrit par Duff et Porter. Notre modèle poursuit une idée de Tanner, et autres, et se révèle peu coûteux. Il est basé sur le concept d'un "frame buffer" virtuel contenant une description de "haut niveau" du dessin et d'une transformation qui lui est associée. Celle-ci transforme cette description en un format RGB servant exclusivement à l'affichage sur un "frame buffer" ordinaire. Nous discuterons des façons d'implémenter ce modèle suivant les possibilités d'exécution en parallèle de différents ordinateurs et "frame buffer" Trois implémentations seront analysées.

The first author's current address is: National Film Board of Canada, Studio A. French Animation, Box 6100, Station A, Montreal, Quebec, Canada H3C 3H5, Tel: (514) 283-9309.

INTRODUCTION

The first computer paint system was written soon after the first frame buffer came into existence. A wide variety of styles and techniques have been implemented since then for a diversity of hardware configurations. This paper will introduce a formal model of a paint system based on an artist's conceptual model similar in many respects to traditional cel animation techniques, but extended to capture new degrees of freedom available to animators through the use of digital computers.

The work reported here is part of a joint project of the Computer Graphics Laboratory and the National Film Board of Canada. In consultation with members of the NFB's French Animation Studio in Montreal, the goal is to build a production-quality paint system. The system has been designed and two prototypes have been implemented. Since the french translation for "paint program" is *palette de couleurs electronique*, the system has been dubbed *Palette*.

Palette is intended not only for painting backgrounds but also for *direct animation*. In direct animation, an image or physical model is changed incrementally and re-photographed to create each successive frame [LAYB79]. In order to reduce the effort required in this labour-intensive process, *Palette* is designed to combine the direct animation potential of a typical paint program with the advantage of *cel animation*: composite images whose component parts are re-usable to save duplication of effort in those parts of the image that remain constant from frame to frame. In cel animation, this is of course achieved by painting each part of the image on a separate sheet of transparent acetate called a "cel."

Palette operates upon "digital cels," that is, RGBA images [PORT84], that may have been painted with the program, digitized from photos or hand-drawn pictures, or produced by other computer graphics rendering techniques. In this respect, its functionality (though by no means its performance) is similar to the *Pixar Compositor* [LEVI84].

The notion of a *virtual frame buffer* with an associated *output transformation* is central to the implementation model. Tanner *et al.* [TANN83a] have described the speed and potential cost advantages of implementing RGB paint programs using a virtual frame buffer as a cache in host memory separate from the hardware frame store used for viewing the image. Frame buffer values need never be read back to the host and the frame buffer hardware has much less stringent speed and depth requirements. [TANN83a] tends to present the concept as a better way of implementing existing applications. The paint system described here is an implementation of that proposal which enjoys the benefits cited, but it demonstrates that another aspect is also important. Separating a "working" description of the image (in a virtual frame

buffer) from the viewing of the image (in a display frame buffer) affords the opportunity of extending the model of a paint system well beyond the set of features supported by today's hardware architectures.

The following sections present details on the artist's conceptual model (a brief "user's manual" for *Palette*), the formal model of a virtual frame buffer that instantiates the artist's conceptual model, the techniques used to implement the augmented brushing styles proposed in the conceptual model, and a brief discussion of some implementation issues that arise when the formal model is mapped onto specific graphics hardware (in this case three specific multi-processor configurations that are being used as prototype implementations of *Palette*).

THE ARTIST'S CONCEPTUAL MODEL

This section provides an overview of *Palette* as seen by its intended user, an artist. The workstation layout (Figure 1) consists of a tablet with stylus as the primary input device, a colour monitor on which the image being rendered is previewed, and an alphanumeric terminal with keyboard. The monitor displays the current image and a set of menus for selecting operations. In the prototype, additional commands and parameter specifications are made through the use of the alphanumeric terminal. A more comprehensive tablet-based menu system is planned for the production system.

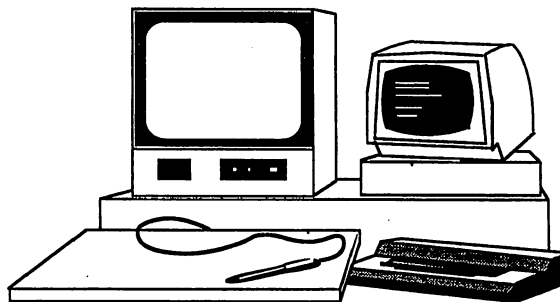


Figure 1. The Workstation Layout for *Palette*.

Central to the conceptual model of *Palette* is the notion that the work surface on which the artist draws or paints is a transparent plane called a *cel*. The term comes from the acetate layers used in animation which were at one time made of "celluloid." In conventional 2 1/2-D cel animation, each frame is created by photographing a stack of cels laid upon opaque background artwork on an animation camera stand [MADS69]. Because cels are transparent except for areas where they have been painted, the photographic process results in a *composite* image. Employing this cel concept in a paint program permits creation of images by composition and provides artwork with the

additional property of variable opacity. The former property facilitates not only computer animation but also a digital form of the "layout and paste-up" process which is fundamental to graphic arts. Figure 2 illustrates the artist's conceptual model of painting with *Palette*.

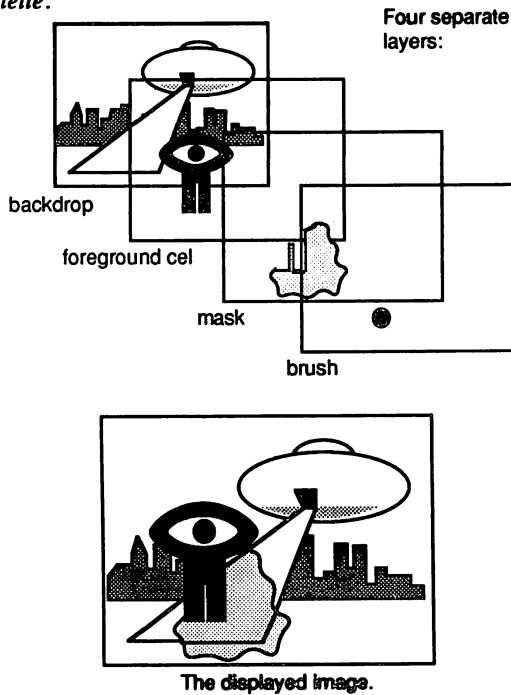


Figure 2. The Artist's Conceptual Model.

The image visible on the monitor during a *Palette* session is the composition of two planes, the *foreground* and the *backdrop*. The foreground has variable opacity while the backdrop is opaque. The foreground level is initially transparent. The artist paints on the foreground level as though it were a "cel." The backdrop may be loaded with a uniform colour or an arbitrary image (including a composition of previously painted cels). The composition of the foreground "cel" level and the backdrop is much like the effect of placing a single cel over a painted background in conventional animation — where the foreground has maximum opacity only it is visible, where the foreground has zero opacity the backdrop is fully visible, and for intermediate opacities the backdrop is partially visible through the foreground.

The prime motivation for the backdrop image is the need for *something* to be visible wherever the foreground is transparent. In addition, while the foreground cel is being painted, the backdrop can be used to hold a reference image for sequence registration (e.g. the previous pose of a cartoon character) or for context (e.g. a background matte painting).

Palette provides only full-colour (24-bit) fully antialiased brushes that have smooth edges and variable opacity determined by brush specifications under the

control of the artist. Essentially, a brush is just another variable-opacity raster image, typically smaller than the image being painted. The painting operation itself is a sequence of applications of the brush to the foreground image using one of a variety of compositing formats to blend the two. Each brush has five orthogonal attributes called *shape*, *stroke*, *colour*, *density*, and *operation*. The artist creates his own brush by assigning attributes to each of the five properties. These attributes determine the effect of applying the brush to the foreground image.

The *shape* property refers to the two-dimensional region of pixels (not necessarily connected) affected by a single imprint of the brush. *Palette* provides a variety of standard antialiased square and circle brush shapes automatically. Alternatively, the user may paint an arbitrary shape to be used as a brush.

The *stroke* property determines the relationship between the motion and pressure applied to the stylus and application of the brush to the cel. When the stroke property has the attribute "stamp," each press of the stylus causes a single composition of the brush with the foreground image. The attribute "repeat stamp" causes a succession of brush composites at a constant rate independent of the speed of stylus motion, thus the gap between brush imprints increases with the speed of the stroke. The "continuous" attribute produces a continuous antialiased stroke without the gaps of the "repeat stamp" stroke. The attribute "straightedge" is similar to "continuous" but always produces a straight stroke between positions indicated by momentary presses of the stylus. A sequence of such strokes is terminated by pressing the stylus at (or very near) the same position twice in succession.

The *colour* property is the set of colours, one for each pixel within the brush shape, which is composited with the foreground image. The brush may be a single colour (the same at each pixel within the shape) or multiple colours.

The *density* property is a weighting function that determines the extent to which a single application of the brush alters the image within its imprint. During painting, density represents the opacity at each pixel within the brush shape. A maximum density causes the colour of the brush to entirely overwrite the cel colour, whereas a zero density leaves the cel unchanged. Intermediate values cause a blending of the brush and image colours using the density as a weighting factor. Densities can be created automatically using constant, Gaussian, or cusp functions centered at the origin of the brush. A Gaussian density function, for example, produces an effect closely resembling conventional airbrush.

The *operation* property specifies one of four modes of brushing: *paint*, *erase*, *mask*, or *mask-erase*. Paint mode uses the brush density to control the blending of the brush with the image.

Erase mode is a unique feature of *Palette* and reveals part of the power of the underlying cel model. In this mode the colour of the brush is ignored, but the density is used to decrease the opacity of the foreground image each time the brush is applied. Thus, where the brush has maximum density, the foreground image becomes absolutely transparent, exposing the backdrop beneath, whereas repeated application of an intermediate density gradually "fades" artwork so that the backdrop becomes increasingly visible through it. A zero density erase brush leaves the foreground unchanged.

Mask mode is analogous to graphic artists' conventional practice of temporarily *masking* areas of artwork by means of paper, masking tape, or *frisket* to protect them from subsequent painting or airbrushing. Again, the colour of the brush is ignored, but the density of the brush determines the permeability of the *mask* associated with each pixel in the foreground image. A maximum-density brush creates an impenetrable mask. During subsequent painting or erasing the effect of a brush will be reduced according to the degree of mask present at each pixel. Masked areas may be set globally visible or invisible by the artist. If visible, presence of a mask is signified by a specified global mask colour.

Mask-erase mode functions similarly to erase mode, but operates on the mask rather than on the foreground cel. It is used to reduce the permeability of a mask or to do away with it entirely.

Palette provides functions for clearing, loading, and saving the foreground, backdrop, and mask portions of an image, for merging the foreground and backdrop images, and for *undoing* the most recent operation applied to the image. Images are stored in a file format that permits a number of other tools in use at Waterloo to be used on images created with *Palette*. One is a general image manipulation package [PAET85] and another is a general compositing package [KLAS85] implementing the full set of operations proposed by Duff and Porter [PORT84].

This section has provided an overview of functions provided in the current implementation of *Palette*. A more complete description of the artist's view of the final system is contained in the functional specification [HIGG83].

THE VIRTUAL FRAME BUFFER

This section discusses the actual implementation of that conceptual model. The key idea which is introduced in *Palette* is the notion of a virtual frame buffer in which the foreground cel and the backdrop, as well as the mask, can be represented. On page 29 of their text [FOLE82], Foley and Van Dam note that the traditional design philosophy of paint programs has been that the image being painted is precisely the image being displayed on the monitor — unlike the rest of

computer graphics, no other representation of the image is maintained. They term this philosophy "what you see is what you get." This approach has advantages in terms of performance but limits paint programs to functionality easily supported in available hardware. Under this philosophy, if its conceptual model is to be taken seriously, implementation of *Palette* would require a substantial investment in custom hardware. In addition to "on the fly" compositing hardware, the frame buffer would require a substantial number of bit planes in order to store the mask, the foreground cel, and the backdrop.

The model presented here breaks with the time-worn "what you see is what you get" approach to paint program design. It is based on a cost-effective approach that separates the painting function from the viewing function by introducing a *virtual frame buffer* in which all painting operations are performed and an *output transformation* that maps the data in the virtual frame buffer into a form suitable for the video-refresh circuitry in a conventional frame buffer. The virtual frame buffer need not be accessible to the video output (in particular, it need be neither dual-ported nor accessible at standard video rates in excess of 100 or even 25 nanoseconds per pixel). Instead, it can be stored in memory that is more readily accessible to stroke rendering routines (either in cpu memory of the host or workstation or in a portion of the physical frame buffer not required for video refresh). Image data in the virtual frame buffer is not subject to the dictates of the video-refresh circuitry and can be stored in whatever format is most efficient for painting algorithms. By employing a higher-level description of the image and formally defining the compositing steps that transform that representation to viewable RGB values placed in the display frame buffer, we adopt the approach that has served the rest of computer graphics so well for so long. We are able to make significant simplification in the programmer's view of the paint program while also freeing ourselves from the limitations of particular display hardware.

A virtual pixel in *Palette* consists of 64 bits of information (Figure 3), 24 bits for each of the foreground and background images, 8 bits for the foreground opacity, 7 bits for the mask, and one additional *contrast flag* that is used to implement temporary feedback images [NEWM79] such as position markers, grid guidelines, and bounding boxes.

Palette's output transformation defines the mapping from these 64 bits to a standard R-G-B representation of each pixel. The first step is to composite the cel and backdrop images according to Wallace's formulation [WALL81]. If the artist has indicated that masking is to be visible, the second step is to check whether the pixel's mask value is non-zero and, if so, to composite the global mask colour over the result of the first step using the mask density value as the opacity. To ensure that visible masking does not overly obscure artwork beneath it, the mask density is

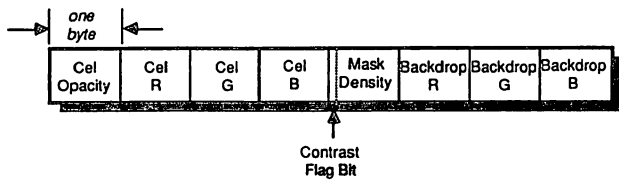


Figure 3. The Virtual Pixel Used in *Palette*.

scaled so that its opacity never exceeds one half. If the pixel's contrast flag bit is set, the final step is to apply a contrasting function to the R-G-B value to map it to a contrasting value that will distinguish the pixel from its neighbors.

Assuming (R,G,B) is an R-G-B pixel value whose components are each in the normalized range [0,1], the contrasting function usually used in raster graphics is $(1,1,1) - (R,G,B)$ [NEWM79]. This is efficiently implemented as complementation of the bit-wise representation for the pixel. This effectively complements the hue of a colour. For cases in which the pixel colour is highly unsaturated and mid-intensity, however, this function produces little apparent change. The worst case is a pixel value of (0.5,0.5,0.5). In the course of formulating our model, Tanner [TANN83b] suggested an alternative function to change a colour by a consistent amount regardless of its original value. The function is $(R,G,B) + (0.5,0.5,0.5)$ modulo 1 and is efficiently implemented by complementing only the high-order bit of each component value.

Palette's cels are based upon the full-colour digital representations of cels and backgrounds for animation described by Wallace [WALL81]. In addition to R-G-B values, Wallace stores an opacity value which is a compact approximation of the edge information at each pixel. Wallace's formula for associative pair-wise composition of cels reduces the total number of compositing steps in a sequence of frames by allowing cels that remain adjacent from frame to frame to be pre-merged. Duff and Porter [PORT84] introduce a compositing algebra in which Wallace's formula is only one of a dozen operations which go beyond what is possible with traditional cels. They call Wallace's "opacity" values "alpha" values and store their images in a different format in which each of the R-G-B components is pre-multiplied by the alpha value. Duff [DUFF85] has extended that model to include a notion of z-depth.

Storing the brush and foreground cel as R-G-B values pre-multiplied by their opacities as recommended by Duff and Porter would require allocating additional bits for each channel in order to avoid severe roundoff error in the course of brush compositing. Our

experiments indicate that at least twelve bits would be needed for each of red, green and blue. Rather than substantially increase the storage required in the virtual frame buffer, the cel R-G-B values are retained in their unscaled form.

The reason originally advanced for storing R-G-B as pre-multiplied values is that the compositing operations can be performed more quickly in that format [PORT84]. *Palette* uses an alternate formulation of compositing due to Hardtke that realizes the same results with almost the same efficiency, while permitting R, G, B, and opacity to be stored as separate 8-bit values [HARD85].

The final issue concerning the virtual frame buffer is the frequency with which the output transformation must be applied. Conceptually, the virtual frame buffer is continually being transformed from its 64-bit internal representation to its 24-bit representation in the physical frame buffer. This is not possible because of the computational bandwidth required. The practical approach is to perform the output transformation at intervals on only those virtual pixels which have been modified. The details of this are very dependent upon the particular architecture upon which *Palette* is implemented. Examples are discussed in Section 5.

BRUSHING TECHNIQUES

The brushes used in *Palette* are relatively complicated objects. For efficiency a *brush record* is maintained that defines the five properties shape, stroke, colour, density, and operation. While stroke and operation are easily preserved as simple scalar values, the shape, colour and density information requires more elaborate data structures. Square and round brushes would be easy to handle, but rather than cater to special cases, brushes are kept in a general linked list data structure whose goal is to save storage and speed brushing by avoiding pixels that are not affected by the brush.

Painting operations within the virtual frame buffer are implemented in a straightforward manner because the foreground cel and the backdrop are stored separately and only the foreground cel or the mask is modified by brushing. To implement the four types of operations, paint, erase, mask, and mask-erase, the brush, the mask, and the cel are treated as images which are combined in various ways using Duff and Porter's compositing algebra. Using the terminology of [PORT84], the paint operation is simply the compositing operation "(brush out mask) over cel." The erase operation is "cel out (brush out mask)." The mask operation is "brush-density over mask" and erase-mask is "mask out brush-density." As has already been mentioned, because *Palette* does not pre-multiply the R-G-B components of its RGBA images by "A" (alpha) as in [PORT84], it uses a slightly different formulation of these compositing operations [HIGG86].

IMPLEMENTATION ISSUES

Practical issues remain to be addressed, such as how and when to update the display frame buffer to reflect modifications to the working copy in the virtual frame buffer. Such questions rely on the particular hardware chosen for the implementation. We thus conclude with a brief overview of three particular architectures on which versions of *Palette* have or will be implemented.

To date, versions of the system have been implemented on two hardware configurations and are planned for a third. The first implementation, a feasibility study, was on a Norpak VDP-1 frame buffer attached via a DMA link to a VAX 11/780 host computer running VMS. A Motorola 68000 microcomputer was attached to the VDP-1 as a dedicated user-programmable display processor. Figure 4 gives a schematic diagram of the system. This equipment is located at the National Research Council of Canada.

In this implementation the entire virtual frame buffer and the undo buffer are located in the host VAX. The VAX is responsible for tablet sampling, all operations on the foreground cel and the backdrop, and the output transformation. The 68000 is responsible for maintaining tracking and writing R-G-B values into the frame buffer. Both processors are programmed in C.

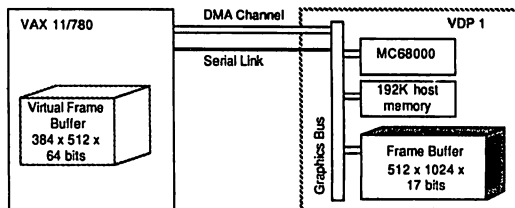


Figure 4. The VAX/VDP-1 Architecture

Because the entire virtual frame buffer is simply an array residing in host memory, many of the bottlenecks associated with traditional host-based paint systems are overcome. Of particular importance is the fact that the VDP-1 is used in a "write-only" manner; the values stored in the hardware frame buffer are never read back during painting. This is fortunate because, like many commercial frame buffers, the VDP-1 does not easily support the operation of read-modify-write on a single pixel. This is the essence of the inner loop of any paint program and it must be made to execute efficiently.

In the VAX/VDP implementation, a stroke is rendered by rubberstamping the brush at each pixel in a straight line between each pair of sampled tablet locations. Fishkin calls this the *Naive* algorithm [FISH84] and notes that the excessive overlap of the brush imprints causes each pixel in the stroke to be

"visited" multiple times by the renderer. The bottleneck in this process is data transfer from the VAX to the VDP-1. In order to minimize the amount of data transferred, the VAX performs the output transformation once on pixels in the "wake" of the brush, that is, pixels that the renderer has finished "visiting." For a given brush shape, the eight possible wake patterns are pre-computed. Each of these define those pixel positions in one imprint of the brush shape that are unaffected by a second imprint offset from the first by one pixel position. (See Figure 5.) After each imprint of the brush shape into the virtual frame buffer in the course of rendering a stroke, the VAX performs the output transformation on the pixels in the wake pattern corresponding to the direction offset between the current imprint and the one before it. The resulting RGB values, the current imprint position, and a number identifying the wake pattern used are all transferred to the graphics processor which writes the pixel values into the frame buffer at the appropriate positions based on its own copy of the specified wake pattern. At the end of the stroke, the full brush shape is used rather than a wake pattern. Figure 5 shows the display buffer updates required for an example stroke. The cross-hatchings indicate the wake pattern that affects each pixel in the stroke.

A noteworthy feature of this implementation is the fact that the VDP-1 at the National Research Council has only seventeen bits per pixel. The actual image displayed on the monitor is generated using the

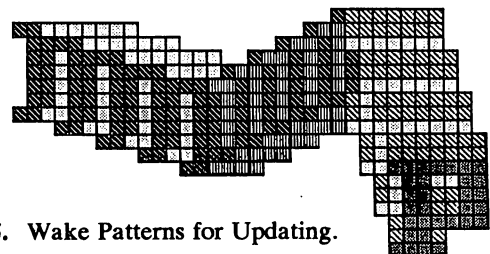
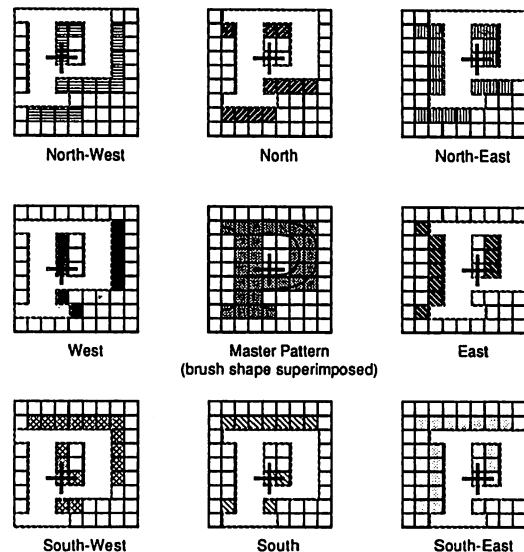


Figure 5. Wake Patterns for Updating.

high five bits of red, the high seven bits of green, and the high four bits of blue. One bit is reserved for tracking feedback. Although this is inadequate for the final image, it is sufficient for viewing the image during its creation. The full-precision version of the image stored in the virtual frame buffer is the useful product.

This first implementation proved the soundness of the artist's conceptual model, but left much to be desired in the way of performance. Reasonable response was precluded by Floating-point compositing code, time-sharing the VAX, virtual memory paging by the operating system, and a 2-millisecond overhead for each system call transferring data to the frame buffer

The second implementation is on an Orca3000 workstation comprising a MC68000 cpu running Unix, a custom bit-slice graphics processor, and a 1024-line 8-bit frame buffer equipped with colour lookup tables [ORCA83]. Figure 6 is a schematic diagram of the system. The workstation used for *Palette* has 4 megabytes of host memory. Its frame buffer memory is only addressable by the graphics processor. The graphics processor is programmed in C using a cross-compiler [GURD85a].

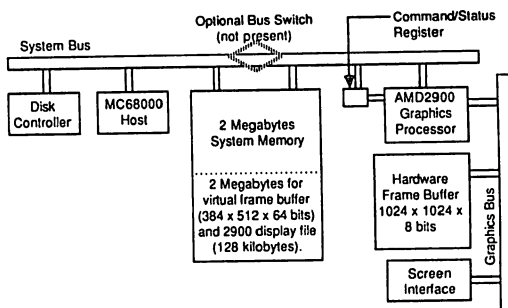


Figure 6. The Orca3000 Architecture.

Virtual memory paging by the operating system is not a concern in this case. The workstation's main memory is ample and its operating system does not support virtual addressing.

An intriguing aspect of the Orca3000 is the manner in which inter-processor communication and data memory for the graphics processor are provided. The graphics processor has an interface to the 68000's system bus and can access any location in host memory directly through the use of *base* and *offset* registers. An additional *command/status register* is used to

communicate with the graphics processor without requiring it to generate contention on the 68000 system bus.

Work is more equitably allocated between the two processors in the Orca implementation because the graphics processor assumes the burden of performing the output transformation. Custom microcode written in C performs the output transformation, the tracking function, and menu-handling on the graphics processor. The 68000 host performs all other functions, including maintenance of the virtual frame buffer. Rather than using the Naive algorithm to render strokes, the 68000 uses a more efficient algorithm which visits each pixel only once. This approach uses Gupta and Sproull's antialiased line rendering algorithm [GUPT81] to look up appropriate values in Fishkin's "Sweep" arrays which contain pre-convolved opacities for a stroke [FISH84].

The "wake" patterns employed in the first implementation of *Palette* are discarded. Frame buffer updates are instead performed by means of a *paging* scheme whereby the virtual frame buffer is divided into rectangular blocks that are marked whenever they are modified. The output transformation is periodically applied to those blocks that have been marked since its last application.

Paging the image to the display frame buffer speeds communication between the two processors and reduces the amount of data transferred. The virtual frame buffer is split into blocks of 16x16 pixels. Each block has a corresponding *dirty bit* that is set by the 68000 host whenever it modifies pixels in that block. After setting dirty bits, the 68000 also sets the value of the command/status register in order to signal the graphics processor which otherwise busy-waits on the register to avoid saturating the 68000's system bus. When alerted, the graphics processor checks the dirty bits to find each block requiring the output transformation. Bits are checked in round-robin fashion to avoid looping on blocks that change frequently to the exclusion of the rest. Before starting the output transformation, the graphics processor resets the block's dirty bit to avoid race conditions with the host. While the 68000 continues painting, the graphics processor accesses the block in host memory directly, performs the output transformation, and writes the resulting pixel values into the display frame buffer.

In the Orca implementation, there is a mismatch in resolution between the 512x512x24-bit image produced by the output transformation described in Section 3 and the 1024x1024x8-bit frame store which must display it. To overcome this problem, an additional step involving *digital halftoning*, is added at the end of the output transformation to map each 24-bit RGB colour to a 2x2 array of 8-bit Orca pixels. In essence three bits of each pixel are allocated to a

halftone version of the red portion of the image, three to the green portion, and two to the blue portion. The halftone patterns are out of phase with each other to avoid *moire* effects and the values in the colourmaps are gamma-corrected for better antialiasing.

Trading spatial-resolution for intensity resolution in this manner is equivalent to an additional two bits in each of the red, green, and blue channels. Thus, the displayed image is effectively 512x512x14 bits. This scheme has worked very well. In side-by-side comparisons with full 24-bit images, differences are difficult to discern at normal viewing distances. Again, a certain amount of discrepancy is acceptable due to the distinction between previewing images during painting and the ultimate resolution required for photographing or video-recording finished artwork. These results support the contention of Tanner, *et al* that the virtual frame buffer approach would permit construction of less expensive "24-bit" painting stations employing hardware frame buffers having fewer than 24 bitplanes.

The third implementation is not yet underway, but is worth considering briefly because it complements the first two approaches. The VAX/VDP-1 implementation performs almost all of the calculations on the host with the frame buffer serving only for viewing. The Orca3000 implementation offloads all of the output transformation to the graphics processor, while maintaining the virtual frame buffer within the host. A proposed implementation for the Adage/Ikonas RDS-3000 frame buffer will move even the virtual frame buffer to the graphics processor while maintaining only the basic tablet routines and high-level control in the "host" 68000 cpu.

The reason for this is that the RDS-3000 supports a full 32-bit pixel and has a 1024x1024 display memory. Because only one fourth of that is needed for the viewing image, the other three-fourths can be used to store the entire virtual frame buffer. A 32-bit custom bit-slice (similar in many respects to Orca3000's 16-bit graphics processor) has sufficient computing power and high-bandwidth access to the display memory that we expect to be able to perform both the basic painting algorithm and the output transformation on the bit-slice without using the 68000 that is attached to the frame buffer.

Because both the 68000 and the bit-slice are programmed in C (the bit-slice has a similar cross-compiler [GURD85b]) we hope to move much of the program from the Orca3000 to the RDS-3000 with little modification.

ACKNOWLEDGEMENTS

The authors wish to thank the National Research Council of Canada for providing access to their VAX/VDP-1 system for the pilot project, to Orcatech for providing the Orca3000 workstation on which the

prototype of *Palette* has been implemented, and to the National Film Board of Canada for supporting the first author during this research. Additional funding was provided by the Natural Sciences and Engineering Research Council of Canada. Special thanks are extended to Marceli Wein of NRC for his suggestions and encouragement.

REFERENCES

- [DUFF85] Duff, T., "Compositing 3-D Rendered Images," Computer Graphics, Vol. 19, No. 3, July, 1985, pp. 41-44.
- [FISH84] Fishkin, K.P., "Algorithms for Brush Movement in Paint Systems," Proceedings of Graphics Interface '84, Ottawa: May 28-June 1, 1984, pp. 9-16.
- [FOLE82] Foley, J.D., Van Dam, A., Fundamentals of Interactive Computer Graphics, Addison Wesley, 1982.
- [GILO85] Giloth, C., Veeder, J., "The Paint Problem," IEEE Computer Graphics and Applications, Vol. 5, No. 7, July, 1985, pp. 66-75.
- [GUPT81] Gupta, S., Sproull, R., "Filtering Edges for Gray Scale Displays," Computer Graphics, Vol. 15, No. 3, July, 1981, pp. 1-5.
- [GURD85a] Gurd, R.P., Microcode C Language Summary - Orcatech Orca3000 Version 1.8, copyright R. Preston Gurd, July, 1985.
- [GURD85b] Gurd, R.P., Microcode C Language Summary - Adage (Ikonas) BPS32 Version 4.8 copyright R. Preston Gurd, July, 1985.
- [HARD85] Hardtke, I. (Master's student in the Department of Computer Science, University of Waterloo), Personal conversation with the author, March, 1985.
- [HIGG83] Higgins, T., Kochanek, D., Langlois, D., Palette de Couleurs Electronique Artist's Guide, Version 1.2, August, 1983 (an internal document of the French Animation Studio of the National Film Board of Canada).
- [HIGG86] Higgins, T., Painting a Cel: Digital Painting in a Virtual RGBA Frame Buffer, Master's Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, 1986.

- [[KLAS85]] Klassen, V. (PhD student in the Department of Computer Science, University of Waterloo), Personal conversation with the author, March, 1985.
- [[LAYB79]] Laybourne, K., The Animation Book, Crown Publishers, New York, 1979.
- [[LEVI84]] Levinthal, A., Porter, T., "Chap - A SIMD Graphics Processor," Computer Graphics, Vol. 18, No. 3, July, 1984, pp. 77-82.
- [[MADS69]] Madsen, R., Animated Film: Concepts, Methods, Uses, Interland, New York, 1969.
- [[NEWM79]] Newman, W.M., Sproull, R.F., Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill, 1979.
- [[ORCA83]] Orca3000 Hardware Reference Manual, Issue 2.0, Orcatech Incorporated, Ottawa, Ontario, December, 1983.
- [[PAET85]] Paeth, A., The IM Toolkit - A Comprehensive Raster Manipulation Package Presented through Design and Example, Master's Thesis, Department of Computer Science, University of Waterloo, Waterloo, Ontario, 1985.
- [[PORT84]] Porter, T., Duff, T., "Compositing Digital Images," Computer Graphics, Vol. 18, No. 3, July, 1984, pp. 253-259.
- [[TANN83a]] Tanner, P., Cowan, W., Wein, M., "Colour Selection, Swath Brushes and Memory Architectures for Paint Systems," Proceedings of Graphics Interface '83, Edmonton: May 9-13, 1983, pp. 171-180.
- [[TANN83b]] Tanner, P., Personal conversation with the author, July, 1983.
- [[WALL81]] Wallace, B.A., "Merging and Transformation of Raster Images for Cartoon Animation," Computer Graphics, Vol. 15, No. 3, July, 1981, pp. 253-262.

DESIGN and EXPERIENCE
with a
GENERALIZED RASTER TOOLKIT

Alan W. Paeth and Kellogg S. Booth

Computer Graphics Laboratory, Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
Tel: (519) 888-4534, E-Mail: AWPaeth%watCGL@Waterloo.CSNet

ABSTRACT

Raster manipulation software is often viewed as an *ad hoc* means to fine-tune the appearance of digital images, or as a means to reformat them to conform to specific hardware requirements. A universally accepted, machine readable, device-independent specification of a raster image is seldom employed. This stands in contrast to the variety of "standards" for higher-level scene representation. We define a general raster "type", which unifies the design of a toolkit of raster-based software. Operations performed by the tools are closed in the sense that operators map objects having the raster type onto new objects having the raster type. This closure encourages a synthesis of function by allowing composition of operators. Sequences of these operators are surprisingly powerful and have wide application.

RÉSUMÉ

Les logiciels de manipulation d'images "raster" sont souvent considérés comme un moyen *ad hoc* d'améliorer l'apparence d'images digitales, ou comme un moyen de les modifier de façon à ce qu'elles se conforment à un appareil spécifique. La représentation universelle d'une image "raster", ne dépendant pas d'une machine particulière, est rarement utilisée; ce qui contraste avec le grand nombre de normes qui existent pour représenter des images de plus haut niveau. Nous définissons un type "raster" qui permet la création d'une série d'outils opérant sur celui-ci. Les outils en question forment un ensemble fermé dans le sens qu'ils opèrent sur des images de type "raster" pour produire des images de type "raster". Cette fermeture permet la création de fonctions par la simple juxtaposition d'opérateurs plus simple. Ces compositions de fonctions se révèlent étonnamment puissantes et ont un vaste domaine d'applications.

Keywords: *bitmap, digital compositing, imaging, raster.*

INTRODUCTION

The ultimate goal of any software system should be the creation of a harmonious set of tools in which each tool embodies a conceptually simple operation. This is true for the case of raster image manipulation, but such a set is not in widespread use. To have generic utility, each tool must operate on an abstract raster type. For instance, a "cropping" tool should trim rasters regardless of their dimension or pixel attributes. Additionally, the tool's output should be, in all cases, a valid raster file so that tools may be composed arbitrarily.

To achieve this, we define a universal file format and implement general raster access routines. With these, the creation and coding of each new tool is greatly simplified, and the proliferation of disposable software can be alleviated. This scenario is also a boon to the user: generic tools imply a simple conceptual model. In some cases, they even suggest new ways of "plumbing" together raster operators. This approach is appealing in an academic/research environment, where creative experimentation is encouraged, but where software maintenance remains on a tight budget.

This paper discusses the design and implementation of a comprehensive raster manipulation system, based on the raster file format, that has been operational for over a year and is the mainstay of raster-based activities within the Computer Graphics Laboratory at the University of Waterloo. In that time, it has completely subsumed the various *ad hoc* raster file formats previously in use and has provided a unifying framework for new research.

OVERVIEW

The toolkit contains programs to support abstract operations (rotation and scaling), as well as interfaces to a number of hardware devices and software systems. These include I/O tools for Adage/Ikonas and Raster Technologies frame buffers, the Apple/Macintosh, and Versatec and Imagen hardcopy printers. In this usual setting, tools model the UNIX text/filter paradigm, whereby the output of any one tool may be piped

directly to the input of the next. This is particularly important when intermediate raster files may be quite large.

The tools are written in standard C, use no assembly code or specialized C packages (such as *yacc*), and have been ported successfully to machines with different word length and severe compiler restrictions. Conditional compiler code is used to represent the specifics of byte order. This allows the system to maintain both a uniform presentation of data for the low-level tool builder and an identical external representation (as a byte stream) for disk files.

Most users are not tool writers, but use the raster tools freely in a conceptual fashion. An artist working on the Macintosh might print bitmap files on the Imagen laser printer, or use them as picture input for the comprehensive Orcatech-based *Palette* [Higg85] colour painting system. Here the user may disregard the different pixel precisions, (lack of) colour, or machine word-lengths, all of which differ for each hardware system. Because the file format has been designed carefully, it has become the format for exchange as well as for archival storage.

We begin by identifying and evaluating the design criteria first for the underlying raster format, then for tools. The paper concludes by demonstrating the synthesis of a new function (digital halftoning) through application of the atomic tools.

BACKGROUND

The widespread availability of digital raster devices has spawned a large progeny of "raster formats", often with no unifying design principles. It is not uncommon for a format to represent a digital "dump" (on external media), patterned after a device's (or program's) internal data structures. It can be argued that this raster format is finely tuned to the hardware characteristics of its respective device, with subsequent advantages in terms of run-time efficiency.

Our findings do not support this argument. Rather than allowing the specifics and availability of hardware or software to drive our choice of design, we make an *a priori* raster file design, and then argue its advantages. We begin by identifying useful design criteria for both the file format and for the general software system in which it is employed.

In contrast to some proposed formats, our design philosophy has been toward a universal file format which is "moderate" in providing sufficient attributes to model any display device, but without any unnecessary or redundant file attributes: it is minimal. This philosophy encourages the construction of tools which embody raster functions in as abstract a setting as possible. As a direct consequence, tools which deal with only a subset of valid raster files will not exist. This approach is a major departure from many other raster systems.

DESIGN OF THE FILE FORMAT

Raster Specification and Operation

No raster specification is present that might be ambiguously interpreted as a raster operation. Thus, "width" and "height" are essential raster specifiers; raster "orientation" is not, because the raster rotation function exists as a tool, and thus is not (by design) a specification. As a consequence, the raster rotation code belongs to a single tool, which aids in software maintenance. This model frees the user from the dilemma over choice of representation and tool application. In previous systems, a custom tool (e.g., a laser output tool) might accept rasters of only a specific orientation, based on speed considerations. Alternately, that output tool might provide a high-speed implementation of rotation independent of a raster rotation tool. In the first case, the user is left with a question of specification to guarantee operation of the printing tool. In the second, the locus of code which provides raster rotation is not well defined.

Some scene representation languages take the opposite extreme. Functional specification is allowed in the most general sense. Here "tools" don't exist *per se*; their function is present in the interpreter which reads a file. An example is the Xerox Interpress standard [Spro81]. Here the general implementation of the file format on a printer implies the existence of supporting code to perform rotation, rendering and all other operations potentially specified by the document. Because this interpreter is monolithic, operations are not free-standing programs. Thus, integration of new software is difficult for a diverse software community.

In our experimental setting, we do not advocate that our raster file allow for "programming" in this sense: we envision situations where one function might be applied to many sets of data, and *vice versa*. Non-radical manipulation of rasters capitalizes heavily on this separation, and we insist on it. In our setting, our well-defined files embody the raster data, and a toolkit of machine-executable files (or UNIX shell scripts) embodies the raster operations.

Pixel Specification

The format provides for the formal specification of a pixel, which allows generic tools (such as crop or rotate) to operate on arbitrary data sets, with independence both from raster dimension and pixel specification. The importance of this should not be underestimated. Historically, formats allow for a maximum of three or four pixel components (often not even within the same file, but as "separates"). Pixel precision is usually taken from a small set, such as one, eight, and twelve bits. Experience has shown that it is impossible to predict *a priori* what or how many attributes comprise a pixel — new models are not to be discouraged. Besides the obvious RGB colour components, traditional data sources often carry multi-

spectral data or Z-depth information. The last few years have brought "alpha" coverage factors and sub-pixel masks to the forefront.

For instance, the Orcatech-based *Palette* system treats pixels as a sixty-four bit quantity, by encoding both foreground and background primary colour information, plus other parameters including masking and transparency, thus modeling artist's use of paint. Our format embraces this experimental system easily. It is worth noting that the system has both a different word length and integer byte order than the original VAX implementation, but this is entirely invisible to the tool creator. Images created by *Palette* may be moved using standard tools (UUCP) to the VAX and rendered on VAX-based graphics engines.

Syntactically, we define pixels as collections of "fields", used to identify the components, in a manner analogous to the record structure type in languages such as Pascal. The pixel attribute is recorded in the file header as a text string. Pixel components consist of an alphabetic identifier and an associated integer which defines the field precision (up to thirty-two bits per component). The identifier is occasionally used externally to specify pixel components to certain software tools. For example, *imextract* merges and extracts pixel components from multiple input files into an output raster, based on the user-specified field set. The precision component is rarely presented to software tools, as the low-level routines allow correct arithmetic operation across files of differing pixel precision (but usually with conforming field names). This is a function unique to our package, and enhances general compositing of files from diverse sources.

Interpretation of Pixel Data

The interpretation of the data fields is at the user's discretion. In many cases, data is taken to span the closed interval $[0..1]$. This interval is closed under multiplication and complementation. The low-level tools provide a data presentation level which returns floating-point values for pixel components on the range $[0..1]$, so the actual field precisions can be kept invisible. This interval is consistent with the design of a number of colour spaces such as RGB, CIELAB and HSB [Smit78], in which the three independent colour axes are placed within the interval $[0.0..1.0]$.

Unfortunately, many software tools in existence wrongly (often implicitly) use the interval $[0..1]$. The latter follows directly when software employs bit shifts to map between pixels with differing numbers of significant bits. In that model, a one bit pixel image (to take the worst case, albeit a very common one), $[0..1]$ allows only the intensity values 0.0 and 0.5. When taken to higher significance, the binary value .1 becomes .10. This system never allows "full-on" to be represented.

A useful mapping has two important properties: *reconstruction* and *representation*. Reconstruction means that data can be mapped into any higher precision, and when subsequently mapped back to the original precision, reconstructs the original data exactly. It is not hard to see that bit shifts are lossless operations and therefore have this useful property. Representation means that pixel data of lower precision can be mapped to a system of higher precision, with the pixel values mapping exactly onto identical intensity values. In general, perfect representation is not possible when moving to higher systems, but it can be achieved in many cases, while providing reconstruction universally.

The proper approach regards the interval as being of length 2^n-1 . In general, our mapping always provides exact values for intensities 0.0 and 1.0, so our interval of representation is the *closed* interval $[0.0..1.0]$. Note that binary (one bit) data in our system represent 0.0 and 1.0 exactly. Adoption of this system means replacing bit shifts (multiplies and divides by 2^m) by general multiplying and dividing. This is not a severe speed penalty. In practice, a scaling table can be constructed and a lookup operation used to find the appropriate mapped value. Our method also provides for reconstruction, because a scale up of one bit provides $2n+1$ new bins, where n existed before, and uniform distribution means that no two values collide.

Exact representation is possible whenever whenever m is a factor of n . To illustrate this, 4 is a factor of 12, so we assert that four bit data has an exact representation in a twelve bit system. To prove that 2^4-1 is a factor of $2^{12}-1$, express them as bit streams: '111111111111' can be divided by '1111' giving '000100010001', or 273. Thus, $4095=15*273$, and the representation for white is still exact. More generally, multiplying any value in the four bit system by 273 yields exact representation in the twelve bit system.

Textual Header

Another departure from many "standard" raster file formats is the exclusive use of case-independent, human-readable text within the header. The use of small "binary" headers with magic word values is still common. Yet in raster files, the header typically constitutes less than 1% of the total storage. The advantage we gain is a parser made common to all user software (and thus is part of the low-level raster primitives). Because our header is minimal, this is a simple task. Direct viewing of the attributes of a file means merely viewing the first few lines of it — no special tool is used.

Because both the header and raster data are represented by a byte stream (giving machine independence), we mandate that no "alignment" specifications to the raster be made — the physical

raster immediately follows the textual header. Experimentation with UNIX-based systems indicates that non-alignment to disk boundaries makes almost no difference to software throughput, particularly where the blocking size on disk transfers is large.

The representation of our header data structure in human-readable ASCII text is a trend increasingly common in good software practice. The design of the highly-successful CIF2.0 by Sproull and Lyon [Hon80] as a VLSI exchange format mandated use of ASCII to allow electronic mailing of design geometries. The format has gained widespread acceptance outside this realm, as it can be implemented easily on machines of differing character representations and word precisions. Sproull previously designed the Xerox AIS [Baud77] raster format (replete with binary header information), and now argues convincingly [Spro83] that this trend toward textual representation should be universally adopted, even where the need for exchange is of secondary importance.

Compact Representation

Archival storage of raster images relies on data compaction. Because we desire a universal format requiring no explicit (de)compression steps, our basic format must provide for compression as part of the pixel specification, and implement this operation as part of the basic access routines.

After two attempts at general data compression, we chose a "compaction" operation, which operates on a level beneath pixel specification, and immediately above the level of physical data movement to external media. Our compaction scheme is a general run length coder, which replaces identical runs of n bytes with $n+1$ bytes of code, representing the original run, plus a count in the range [1..256]. We choose the term "compaction" over "compression", as the operation may take place without regard to pixel boundaries. Early experiments indicate this may have value where images containing data that is cyclic across a scan-line (half-toned images, stipple patterns) are to be encoded. The compression size can then be set to span a collection of adjacent pixels.

General and Special Cases

The raster proper is encoded in a manner which maximizes speed of raster (un)packing by aligning pixel groups onto regular boundaries. Although the specifics are detailed, this underlying code guarantees packing efficiencies of more than 84% for pixel sizes up to 12 bits, approaching 100% for arbitrarily large pixels. This design choice minimizes overhead in the data extraction loop, as the shift and mask values are constant over the data set.

The criteria set forth above allow "special cases" to fall out directly from the more general specification, without special caveats being coded in. This is intentional. For instance, the external representation of an 640x480 size raster of twenty-four bit RGB pixel values is quite simple: a textual header, followed by 640*480*3 bytes of data, arranged in R,G,B order, by scan-line, without any padding. Although we don't advocate that tools write rasters independently of the low-level software which defines the header specification, it does indicate the simplicity and generality of our approach. For instance, a videotex station could dump out a hard-coded header string, followed by a byte dump of its screen contents thus creating a well-formed "canonical" raster format file.

TOOL DESIGN

General Philosophy

Brooks' findings [Broo75] show that as a rule a long-lived systems consist of software which outlives the intention of its original use. Because we cannot anticipate the user's ultimate needs or goals with the raster tools, we should choose to craft each tool into an atomic, composable function with no implicit assumptions of the user's intentions. From this "metaobjective", a number of practical considerations immediately become clear.

Consistency of Design

Overall design consistency leads to tremendous ease of use for both implementor and user. In particular, the time spent in learning the tools used for simple operations becomes proportional to the user's objectives; what little "start-up overhead" exists is common to all tools, and need not be relearned. Similarly, the tool designer can fashion a new tool based on the existing package, thus implicitly inheriting uniformity of the user interface (such as commonly used command line switches) and operation.

As an example of consistency, all software tools dealing with the concept of an axis-aligned rectangle specify this entity in terms of origin and size, not as diagonally opposed corners. In contrast, corner-based specifications leave the ambiguity of semi-open intervals for the user to resolve. For instance, the corner specification model might describe a 512x512 display as spanning the region (0,0) to (511,511), whereas our model describes the display as a window of size (512,512) with origin location (0,0). Thus, we remove the burden of potential "off by one" errors; in fact the casual user will probably be unaware that any ambiguity could exist. This "correctness by design" is a very powerful concept in implicitly steering the user along a correct path of conceptualization.

Minimal Atomic Set

The tools are atomic, composable functions which deal with raster data in the most abstract way conceivable for each respective function. They strongly resemble Guibas's concept of a bit map calculus [Guib82] with the accompanying language MUMBLE. Our implementation provides for pixels of arbitrary precision, as do his; our "language" consists of UNIX commands in which tools play the part of keywords to perform manipulations on the data. A compiler for a large subset of MUMBLE using the toolkit as "machine code" would be a straightforward exercise.

Just as computer languages advocate a small number of composable keyword constructs, we encourage the user to synthesize function from tools already within the kit. When this fails, he should seek the most general tool necessary to extend the coverage of the tool set to contain this specific operation. Besides allowing for a new function with the least amount of new software, a minimal addition to the toolkit can be very revealing to the deep structure of the problem.

THE TOOLS

Although space does not permit a description of each tool, we may summarize the operation of the toolkit. It is useful to characterize classes of tools by common operation. These form our taxonomy.

Storage Considerations

Because tools are composable, they operate on data presented serially. However, some tools require internal raster storage to perform their intended operation. We classify these as "level 0" (constant pixel storage needed), "level 1" (constant scan line storage needed), and "level 2" (arbitrary storage). In each case, these are worst-case raster storage requirements. A generous number of tools belong to classes 0 and 1. Sequences of operators may therefore manipulate images on secondary storage whose size exceeds system main memory.

Input/Output Characterization

Because tools are operators, we may classify them as "unary", "binary" or "tertiary" tools, based on the number of simultaneous inputs used. Two additional classes: "source" and "drain" represent tools which interface between the toolkit universe and some other means of representation. These include display output tools, text input tools and pattern generators. With the exception of drain tools (these typically render images), a tool will have one "standard" output in the form of a universal image file. This class characterization is formally specified in the source code of each software tool, thereby activating library routines common to all tools within this class. Such code governs the number of expected input files and enables command line switches generic to that class.

Other Characterizations

A final characterization is whether a tool preserves pixel integrity. Most tools do, and this is important when a tool is used to manipulate non-intensity pixel fields, particularly when the tool is used in settings far removed from traditional imaging applications. Cropping can be performed on data which contains Z (depth) information, but a low-pass filtering of such data is not intuitive because the latter does not preserve pixel integrity. At present, few tools which violate pixel atomicity exist. Pixel-preserving functions fit in well with our design philosophy of generic tools.

TOOLS BY EXAMPLE — HALFTONING

The following examples demonstrate a series of experiments used to perform digital halftoning (the creation of bi-level images) from high resolution sources. The presentation is an idealized "lab session", but it is also a recapitulation of the historical development of the tools.

Experiment #1 — Plate #1

We begin by halftoning through simple thresholding. We envision thresholding as a binary tool which does a test for magnitude of its two inputs. The source tool `1mconst` is used to provide a reference level for the secondary input. Generally, thresholding can be modeled as a subtract operation, with a subsequent test to map `x>0` values into white, else black. This last function already exists as `1mtomask`. We thus perform the test `a>b` as `(a-b)>0` stepwise on the image file containing a "milkdrop":

```
1mconst -d milkdrop.1m -v 128 |  
  1msubtract milkdrop | 1mtomask >out
```

Experiment #2 — Plate #2

Thresholding to a uniform, constant value produces poor (as expected) images, so we write a program `1mhalftone` to do ordered dithering, comparing input pixels against a cyclic, periodic set of dynamic threshold points to vary the thresholding [Jarv76]. The program is not clean: the 4x4 matrix of threshold weights is hard coded, and the software must permute the array internally to conform to the arbitrary widths and heights of the input file.

```
1mhalftone milkdrop.1m >out
```

Experiment #3

The halftone results look good, but we need avenues of further exploration. The permuted internal weight table replication code is in fact a "tile" operation first conceived for use with much larger images. We write `1mtile`. As a bonus, the tiler is fast, and includes offset switches to be fully general. These correspond to phase shifts in the halftoning dot, a desirable feature in colour digital halftoning, in which

halftone screens for successive colour separations are staggered with respect to previous screens. The threshold table is now recoded as the file kernel.im.

```
imtile kernel.im -w 128 -h 128 |
  imsubtract milkdrop | intomask >out
```

Experiment #4

The 4x4 kernel, now represented as kernel.im in Experiment #3 was cumbersome to make. We had also planned software which would do a "text dump" of raster files; here we wish to do the opposite: convert the "dump" into a raster representation. Realizing that the scope of this software is more than merely one of diagnostic service, we write both imtabin and imtabout. The hard coded ASCII constants of imhalftone have now found a niche. One can envision a standard tool sequence (e.g. a UNIX shell script) to halftone arbitrary images against a textually encoded table of weights.

```
imtabin -w 4 -h 4 -p n8 >kernel.im
240 176 80 208
96 16 48 128
160 32 0 64
192 112 144 224
^D
```

Experiment #5 - Plate #3

The typing of constants in Experiment #4 suggests a mechanized means to generate random numbers, and we are curious to see the appearance of such output. The creation of an array of random numbers (not specific to halftoning) is all that is needed: the other code is already in place. We rewrite of copy of imconst which substitutes random values for constants. The -default switch in our example borrows the dimensions and pixel specification of milkdrop.im, so that imrandom can produce conforming output. The output shows superimposed high-frequency noise [Robe62], resembling the grain in film emulsions "pushed" too far during development.

```
imrandom -d milkdrop |
  imsubtract milkdrop | intomask >out
```

Experiment #6 - Plate #4

The results inspire the use of a thresholds with Gaussian distribution. We recall that the "coin tossing" method [Kalb79] generates such sets by averaging small sequences of evenly distributed numbers. This requires binary operators other than subtract (such as average and sum), so we extend the scope of imsubtract. The tool is renamed imaop because it now allows for arbitrary arithmetic operations from two input sources. We also rediscover that rerunning imrandom generates identical values, so we employ imcrop to give us a set of different random numbers.

```
imrandom -d milkdrop -h 512 >master
imcrop master -y 0 -h 128 >m1
imcrop master -y 128 -h 128 >m2
imcrop master -y 256 -h 128 >m3
imcrop master -y 384 -h 128 >m4
imaop m1 m2 -op aver >t1
imaop m3 m4 -op aver >t2
imaop t1 t2 -op aver >gauss
imaop milkdrop gauss -op sub | intomask >out
```

Experiment #7

The brevity of most command lines is pleasing, but intomask is ever-present. Because it is a unary operator immediately following imaop in each case, we extend imaop to include a thresh function. The code integration is trivial: three additional lines and a new switch statement label. As a bonus, the thresholding works "automatically" for colour files - a feature unanticipated at the outset. Thus, imhalftone has now been made obsolete.

```
imaop milkdrop gauss -op thresh >out
```

Conclusions

The evolution of the imaging software demonstrates a few important principles. For one, generality of function allowed a means to verify hypotheses, without any programs having to be written. As a clearer understanding of the desired goal emerged, specific tools were crafted. For instance, we created random numbers with Gaussian distribution by a simple synthesis of operation, thus allowing the user a glimpse into their properties. Should this become a desirable feature to support in general, -gauss or -seed switches may be added to imrandom, but for current applications this is unnecessary.

These examples show that when a new operation is sought, it can often be melded into the function of a tool in existence, thus widening the scope of operation for the original tool. This creates a tremendous synergy of function. By studying why more than one path toward a goal exists, we can both pare down what constitutes a minimal set, and simultaneously get new insights into the deep structure of the problem.

Acknowledgements

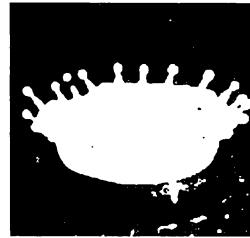
The complete raster toolkit has been distributed on a limited basis to a number of installations. A more formal release is in preparation, pending the completion of a comprehensive technical report.

The authors wish to thank the many members of the Computer Graphics Laboratory who commented on this research, especially Michael W. Herman with whom many discussions were held during the preliminary stages of the design. The research reported here was supported by the Natural Sciences and Engineering Research Council of Canada under a variety of grants. The first author was partially supported by a University of Waterloo bursary.

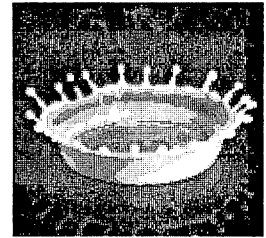
REFERENCES

- [[Baud77]] Baudelaire, P., Israel, J., Sproull, R. "Array of Intensity Samples - AIS" Xerox PARC Internal Report, February 1977 (Rev. by K. Knox, 1980)
- [[Broo75]] Brooks, F. P. Jr. *The Mythical Man-Month - Essays on Software Engineering* Addison-Wesley, Reading, MA (1975) pp. 120.
- [[Floy75]] Floyd, R. W., Steinberg, L. "An Adaptive Algorithm for Spatial Gray Scale" *Society Inf. Displays Int. Symp. Digest of Technical Papers* (1975) pp. 36.
- [[Guib82]] Guibas, L., Stolfi, J. A Language for Bitmap Manipulation" *ACM Transactions on Graphics* 1(3) July 1982, pp. 191-214.
- [[Higg85]] Higgins, T. M. "A Cel-Based Model for Paint Systems" Master's Thesis, University of Waterloo, Waterloo Ontario, May, 1986
- [[Hon80]] Hon, R. W., Séquin, C. H. "A Guide to LSI Implementation" Xerox PARC Bluebook SSL-79-7 (2nd edition January 1980).
- [[Jarv76]] Jarvis, J. F., Judice, N., Ninke, W. H. "A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays" *Computer Graphics and Image Processing* 5(1) March 1976, pp. 13-40.
- [[Kalb79]] Kalbfleish, J. G. *Probability and Statistical Inference* (Vol I), Springer-Verlag 1979, Sec 6.7 pp. 234-239.
- [[Robe62]] Roberts, L. G. "Picture Coding using Pseudo-Random Noise" *IRE Trans. Info. Theory* IT-8 (February 1962) pp. 145.
- [[Smit78]] Smith, A. R. "Color Gamut Transform Pairs" *ACM Computer Graphics (SIGGRAPH '78)* 12(3), August 1978, pp. 12-19.
- [[Spro81]] Sproull, R. F., Lampson, R., Warnock, J., Reid, B. "Interpress: A Standard for Communicating and Storing Print Graphics" Xerox PARC Private Document ISL-81-1 (Subsequently released and made available by Xerox Corp).
- [[Spro83]] Sproull, R. F. *Private Communication*, Xerox PARC, May, 1983.

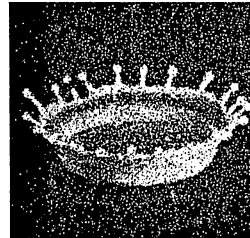
PLATES



1. Simple Threshold



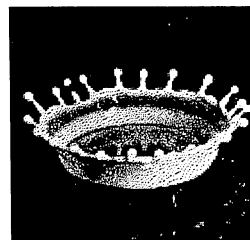
2. Ordered Dithering



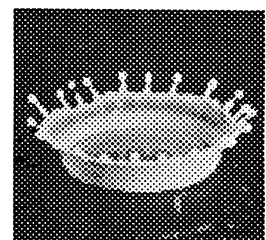
3. Linear Random



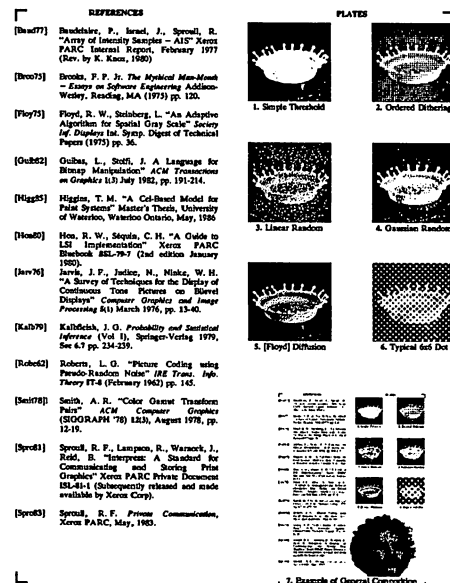
4. Gaussian Random



5. [Floy75] Diffusion



6. Typical 6x6 Dot



7. Example of General Composition

Graphics Tools in Adagio, A Robotics Multitasking Multiprocessor Workstation

Stephen A. MacKay
Peter P. Tanner[†]

Laboratory for Intelligent Systems
National Research Council of Canada
Ottawa, Ontario K1A 0R8 Canada.

Abstract

The development of Adagio, a robotics simulation workstation, has involved the implementation of several techniques unique to the system. Based on the message-passing, multitasking multiprocessor realtime operating system Harmony, Adagio is programmed using a large number of cooperating tasks. Several techniques are based on the concept of a server, a task that is alone responsible for governing a scarce resource. The Graphics Server task, the Data Structure Server task, and the Tracker Server task are responsible for the management of the frame buffer, the 3D geometric data structure and the screen tracker, respectively. Each of these servers is then a separate tool necessary for the implementation of the whole system. Each runs in parallel with other tasks and can handle requests for service from any task.

Résumé

Le développement d'Adagio, station de travail dédié à la simulation de robots, a conduit à l'implémentation de techniques originale. Basé sur le système d'opération de temps réel, multi-tâche, multi-processeur Harmony (lui-même basé sur le transfert de messages), Adagio a été programmé en utilisant un grand nombre de tâches coopérants entre-elles. Plusieurs de ces techniques sont basées sur le concept de serveur (le serveur est la seule tâche responsable d'une certaine ressource). Les tâches "Graphics Server", "Data Structure Server", et "Tracker Server" gèrent respectivement la mémoire d'image, la base de données géométriques 3D, et le curseur d'écran. Chacun de ces serveurs est donc un outil séparé, nécessaire à l'implémentation du système complet. Chacun s'exécute en parallèle avec d'autres tâches et peut gérer des demandes venant d'autres tâches.

Keywords: robot simulation, realtime, multitasking, multiprocessing, message passing, server task, frame buffer, windows, 3D geometric data, user interfaces, screen tracker.

[†] current address: Computer Graphics Laboratory
University of Waterloo
Waterloo, Ontario N2L 3G1 Canada.

NRC number : 25461

Introduction

Adagio is a robotics simulation workstation currently under development at the National Research Council. The workstation, when completed, will give the user the capability of creating and manipulating 3D objects in a robot environment, of specifying the robot task, and of viewing the results of a robot simulation. As such it will certainly be usable for many other applications that can make use of a 3D window-based near realtime display with extensive interaction capabilities.

The use of the Harmony operating system, a multitasking multiprocessor realtime message-passing system, as a base, has led to different approaches to the software architecture of an interactive graphics system. This paper discusses several of these approaches.

Three servers, a Graphics Server, a Data Structure Server, and a Tracker Server follow an idea common in multitasking systems; i.e. each is solely responsible for a specific scarce resource. The Graphics Server is charged with the maintenance of the frame buffer, while the Data Structure Server maintains the 3D geometric representation of the robot and its environment. The Tracker Server communicates with the Tablet Server to provide continuous tablet tracker echoes. It is particularly well suited for a multiwindow system and provides richer user feedback than is currently available on most systems.

Adagio Overview

Goals

Adagio [Tann85b] is a workstation being developed to support research in intelligent robotics. It is intended to provide a simulation facility for studies in the off-line programming of sensor-based robots. The functional requirements of providing the user with a view of the current status of the robot in its environment with near realtime updating (i.e. 5-30 frames per second), and providing for rich interactive dialogues for experiments in interactive graphics-based robot programming, led to the use of a powerful frame buffer display (in our case, an Adage 3000 graphics system) with a window-based user interface. Special software for the Adage 3000 bitslice microprocessor has been written to support multiwindow near realtime line-drawing and polygon faceted 3D renderings of a single scene.

Another goal has been to take advantage of the multitasking inherent in the Harmony operating system to improve various aspects of interactive graphics systems. This has resulted in implementing a highly parallel base for user interaction [Tann85a], a switchboard input model [Tann86], and the various tools described in this paper.

Harmony

The Adagio system design is influenced by the architecture of Harmony, a multitasking realtime operating system with rapid inter-task message passing developed at the National Research Council of Canada [Gent85]. A few details of its properties are given here in order to aid in the appreciation of the multitask design presented in the paper.

Programs written for a Harmony-based system tend to be implemented as a set of many small tasks. A task is often used as one would use a subroutine in a conventional operating system, except that an instance of the task must be explicitly created, and once it has been created it executes independently of, and in parallel with, the task that created it. Task primitives are relatively cheap, message communication takes little time, (a send-receive-reply sequence takes about one millisecond), and the creation and destruction of tasks are inexpensive. Tasks can be created and destroyed as needed, and often are quite small with short lifetimes.

The communication and synchronization of tasks, used extensively throughout this workstation design, is based on the send-receive-reply mechanism provided by Harmony. A task may send information or a request for information to another task by issuing the `_Send` command, passing a variable-length message. If the recipient task is alive, the sending task then blocks until a reply arrives from the recipient or the recipient is destroyed. A task receives a message by issuing a `_Receive` command, which blocks until a message is received, or a non-blocking `_Try_receive` command. In either case, the ID of the sending task is returned to the recipient task (0 if no message was waiting in the case of `_Try_receive`) along with a copy of the message that was sent. The `_Receive` or `_Try_receive` command may specify that messages are to be received from a specific task, or from any task. A task that has received a message from another task may reply to the sending task with the `_Reply` command. The `_Reply` command unblocks the sending task and causes a variable-length message to be replied to it. A task need not reply to a sending task immediately, replies may be issued at any time and in any order necessary to achieve synchronization.

The send-receive-reply paradigm also encourages the use of *server* tasks, based on the *administrator* concept [Gent81], to perform various duties for other *client* tasks, such as the managing of scarce resources. A typical server never sends messages, it only receives and replies to requests. It often will have one or more *worker* tasks doing the time consuming work so that the server can respond to requests as quickly as possible. Because most servers do not send messages, two servers that must communicate usually do so through a *courier* task created for the purpose. A courier alternates between sending a request for information to one task and sending the resulting information to the other. The Graphics Server, the Data Structure Server and the Tracker

Server described below are three examples of servers used in Adagio.

Hardware Configuration

The Harmony operating system is particularly suited for running on a multiprocessor. Such a machine, a Chorus multiprocessor, consists of three single board computers, using Motorola 68000 processors on a Multibus backplane. Providing the graphics processing for the workstation is an Adage/Ikonas 3000 graphics system, a powerful frame buffer display with a 32-bit bitslice processor, a single 68000 also running Harmony, and an image memory 1024 by 1024 by 24 bits (512 by 512 visible).

Interaction Model

Adagio's design is based on the concept of a *switchboard* [Tann86]. The Switchboard, shown in Figure 1, is a server that corresponds with a number of input device servers, through couriers, and a number of client tasks that make use of the values from these devices. The client tasks request input from the Switchboard, which in turn routes to these tasks input from devices to which they are connected. A flow of messages is thus established going back and forth between the producers and the consumers of input.

Graphics Server

The Graphics Server, shown in Figure 2, replaces the graphics subroutine support package traditionally used in a single-task graphics program. Running as an independent task with the role of managing the frame buffer and the Adage bitslice processor, this server handles three types of request messages - window manipulation messages, 2D graphics messages, and 3D graphics messages.

Screen Windows

Screen windows in Adagio are implemented in a manner different from those of many other window based systems [Tann86]. All windows are tightly coupled, assisting in the single job of creating and manipulating a data structure defining the robot, its environment, and the actions of the robot. A tiled window approach is used to simplify rapid screen updates. The system supports the display of two different types of windows, 2D and 3D. Each 2D window, used for displaying text and 2D symbolic graphics, has its own associated task responsible for all activities in the

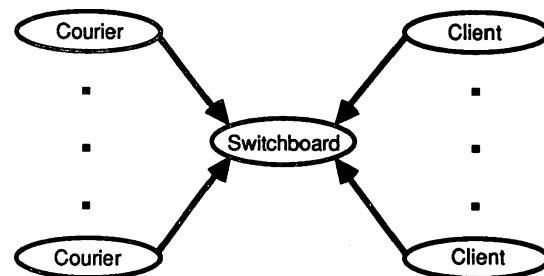


Figure 1. The Switchboard task. (Note that arrow-heads point away from the task making the request.)

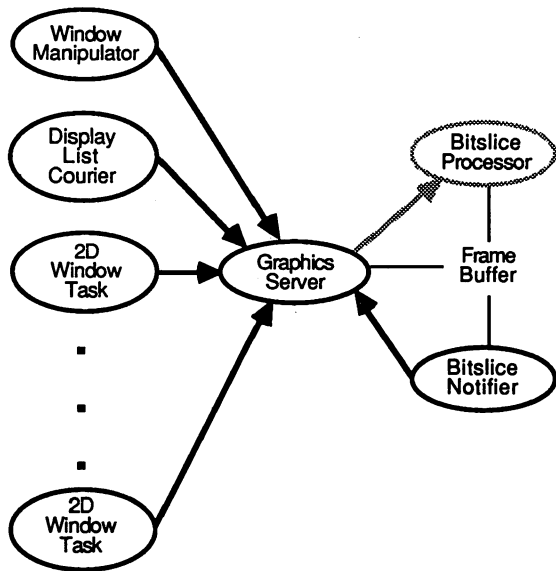


Figure 2. The Graphics Server.

window. All 3D windows, used for displaying different views of the robot and its environment, are controlled by a single task, the Data Structure Server, through the Display List Courier, because each window is a different representation of the same robot data structure.

The Graphics Server is responsible for displaying the contents of both types of windows. As well, it must change the way in which the windows are displayed on the screen in response to window manipulation messages from the Window Manipulator task that request the creation and modification of the window structures maintained by the Graphics Server.

2D Graphics

The 2D graphics messages request text or symbolic output to be directed to the screen window associated with the requesting task. The Graphics Server is responsible for translating from the virtual coordinate system of the requestor to the screen coordinates of its window. If required, the messages to a window may be stored and then reinterpreted if a window is modified in size, or if a picture segment that potentially blocks part of the window is moved. This is different from many other tiled window systems that require the application running in the window to become involved with the redraw of a window that has been changed in size.

3D Graphics

The 3D graphics messages are in the form of commands for the Graphics Server to modify the display list. The display list is in turn interpreted by the Adage bitslice processor to render the 3D image into the frame buffer [Loo86]. While the microcode running in the bitslice processor is not strictly speaking a Harmony task, and shared memory is used for communication, rather than the send-receive-reply message passing primitives, it can still be viewed externally like any other independent Adagio task.

The Graphics Server acts as an *agent* task, as described in Plebon and Booth [Pleb82], for the task running in the bitslice processor, providing an interface to the other processor and managing the communication between and shared resources of the two processors. Because requests for the bitslice processor to update the display list are buffered by the Graphics Server until the bitslice is ready to begin another update, client tasks requesting the services of the Graphics Server do not remain blocked for long.

Currently, the bitslice processor cannot interrupt the 68000 to signal completion of a screen update, so a worker task, the Bitslice Notifier, is created by the Graphics Server with the sole responsibility of informing the Graphics Server when the task running in the bitslice processor has finished. It sleeps most of the time, occasionally waking to poll a flag and sending a notification message to the Graphics Server when the screen update is finished. The incoming requests to modify the display list that the Graphics Server had been buffering are then satisfied, the bitslice processor is released to update the display, and the Graphics Server replies to the Bitslice Notifier thus releasing it to run again.

The display list supports multiple views of a single environment where these views may differ in their viewpoints as well as their display parameters. Modifying display parameters from one 3D window to another permits, for example, the posting of shaded images of the robot in one window and a simple stick figure of only the axes of rotation of each of the joints in another - both rendered from the same display list. Messages, resulting from user actions or simulation of robot activity, can request the rotation or translation of robot links. These require only a change of the appropriate transformation matrices in the display list and a signal to the bitslice processor to re-render the image.

Data Structure Server

With the multitasking approach to the workstation, it is quite feasible that more than one task would wish to update or query the structure representing the 3D geometry of the robot and its environment. To prevent corruption of the data by concurrent accesses, the data structure is known only to a single task, the Data Structure Server, shown in Figure 3. All requests for information from the data structure, for data structure updates, and for manipulation of 3D windows are fielded by the Data Structure Server. Any update that requires a modification of the screen image is forwarded by the Data Structure Server to the Graphics Server, through the Display List Courier to avoid blocking for a long time. When a major portion of the graphics data structure must be sent to the Graphics Server for the creation of the display list, a pointer to the structure is sent, thus making the structure temporarily known to another task. However, until the Display List Courier reports the completion of the screen update, the Data Structure Server will satisfy only 3D data information requests. Requests to change the data will be held until it is safe to do so.

Tracker Server

Management of user feedback is an important element in any interactive system. The user must often keep in touch with several activities simultaneously. He must know what actions are available to him and what the system is doing, and he needs reassurances that all is progressing as it should.

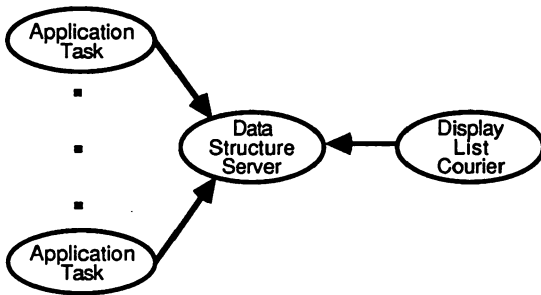


Figure 3. The Data Structure Server.

One element of system feedback is the *screen cursor* or *tracker*. Always displayed at or near the user's centre of attention, a well designed tracker that changes in response to system activity is a powerful indicator of the state of the world. Tilbrook [Tilb76], [Baec80] shows that a tracker can provide much more information than simply the X, Y position of a locator device. Plebon and Booth [Pleb82, pp. 26-28] provides additional information on the use of trackers as feedback and gives further references.

The term *tracker* is used for describing the feedback showing the current X, Y position of a locator device (mouse, tablet, joystick, etc.), because it is simple, it correctly describes the activity (tracking a locator device), and it avoids ambiguity. Although *cursor* is the most widely used term for locator feedback, it also has been associated with the flashing bar, line or box found on most alpha-numeric terminals to prompt for text input, with the hand-held physical device used for positioning on a graphics tablet (puck), and even with the the cross-hair wires at the tip of the puck.

The Locator Model

The Tracker Server assumes a locator model such as the one provided by the Adagio Tablet Server [Tann85b]. In addition to X, Y coordinates, the Tablet Server returns a *window*, or ID of a predefined region on the tablet surface; and a *status* of the pointing device. The status is dependent on both the current state of the tablet and the state during the previous read operation, and may be one of: UP/UP, UP/NEAR, NEAR/NEAR, NEAR/DOWN, DOWN/DOWN, DOWN/NEAR, or NEAR/UP.

Icons

In an environment where many activities may be controlled by a single input device, in particular a window based system, the tracker must be able to provide feedback indicating the action being performed. Having the tracker displayed as one of several possible graphical *icons*, or pictorial symbols, can help accomplish this. These icons, drawn from a set of icons defined for the system, (not all of which are used for trackers), enrich the interaction because different ones may be used to indicate the window where the tracker currently resides, the state of the task for that window, and perhaps the button most recently pushed. Figure 4 shows some currently used icons.

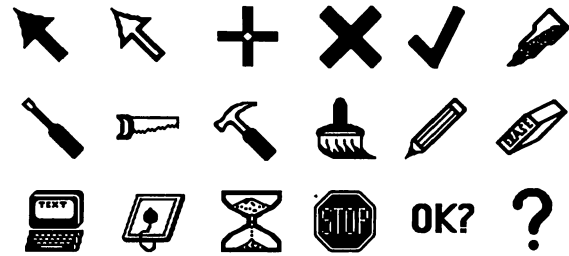


Figure 4. Some Adagio icons.

The Server Task

The Adagio Tracker Server, shown in Figure 5, offers advantages over some other approaches to tracker management. For example, the University of Waterloo Paint program [Pleb82] [Beac82] also uses a single tracking task, but it is a small worker task, created whenever interaction is permitted and destroyed when it is not. Paint provides little control over icons, they are compiled into the program. The worker task has only three kinds of trackers and relies on a few global variables for information about the trackers.

The Tracker Server, taking advantage of the powerful Harmony server model, maintains the data structure of all icons in Adagio, allowing icons to be *designated* or *added*, to be removed, or to be modified. Any icon may be *invoked* or made known to the graphics hardware as the current tracker. The Tracker Server can *bind* an icon with a particular screen window, the status of a locator device and the button value of the locator (together called an *icon bundle*).

Icon bundles allow different trackers to be used for different states of the system so that the tracker best reflects the user's current activity. Bundles can be stacked, so that old bundles can be remembered if a temporary action needs to use the same window, status and button information as a previous action. This temporary action could be, for example, the changing of the tracker icon to Tilbrook's Buddha or Macintosh's wristwatch indicating that the window is busy, or when, through some mode selection, a different tracker is required to indicate the new mode. Bundles subsequently can be removed to restore the previous bundle or to break the icon-window-status-button association.

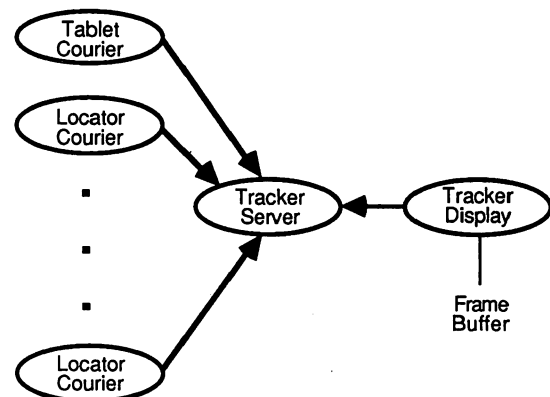


Figure 5. The Tracker Server.

The Tracker Server allows trackers to be positioned in two ways, either by bundle or by naming the icon. Also, the server allows the tracker to be turned on or off as needed. The Tracker Server is device independent, it relies on the device dependent Tracker Display worker task to position the tracker on the screen.

Complete System

Figure 6 shows many of the tasks discussed in this paper. (Currently one may have 40 to 50 tasks concurrently active.) It is obvious from this figure that rapid message passing is crucial if the user is to see the results of his actions reflected in the image on the screen in a reasonable amount of time. Harmony does indeed provide such a basis, and its existence has encouraged the experimental techniques developed in this project.

Conclusion

The paper has described a number of servers currently in use in Adagio, each responsible for a certain resource. The use of these servers offers an abstraction between the specific features of the resource and the users or clients of the resource. The Graphics Server offers device independence as do many available graphics subroutine packages. However the Data Structure Server extends the idea of independence into the realm of data structures. The Tracker Server hides from its clients the peculiarities of the particular hardware tracker.

A second advantage of the server approach, coupled with the use of clients, couriers, and notifiers, is the degree to which it makes it simple and natural for people to structure and code programs for multiple tasks and multiple processors. Resulting programs exhibit a high degree of parallelism, making possible the efficient use of multiprocessors - a necessity considering the direction of hardware development.

A high degree of modularity also results from the use of servers. The server model encourages the building of tools that are almost completely self-contained. All details concerning a resource can be easily encapsulated in a single unit. The interface is usually as simple as sending one of a predefined set of messages to the server and expecting one of a small set of predefined replies in return.

Multitasking models of programming for interactive systems are not only useful for reasons of computing efficiency, but provide a far more appropriate base for computer-human interaction. Contrary to the belief of many system designers [Kern84], a human is not a file to be read. The narrow "user is a file" belief has led to the traditional interactive dialogue where the human uses a specific device in reaction to the systems commands. A multitasking system, made more simple to program using tools such as servers, can easily give far more control to the user by providing him with a variety of tools waiting to serve him.

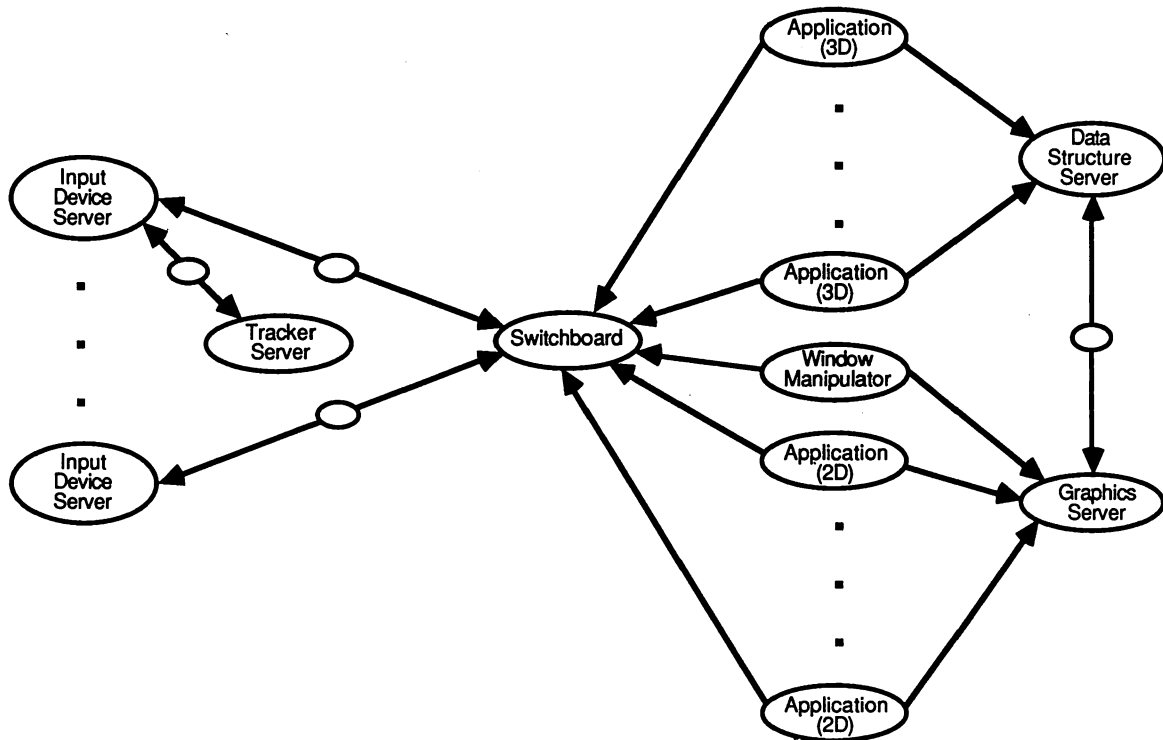


Figure 6. A typical configuration of the complete system. Only major tasks are shown, worker tasks, for example, are omitted. Significant couriers are shown as small unlabelled ovals.

Bibliography

- [Baec80] R.M. Baecker, D.M. Tilbrook, M.I. Tuori, D. McFarland, "Newswhole," SIGGRAPH video review #1, 1980.
- [Beac82] R.J. Beach, J.C. Beatty, K.S. Booth, E.L. Fiume, and D.A. Plebon, "The message is the medium: Multiprocess structuring of an interactive paint program," *Computer Graphics*, vol. 16, no. 3, pp. 277-287, July 1982.
- [Gent81] W.M. Gentleman, "Message passing between sequential processes: the reply primitive and the administrator concept," *Software Pract. & Exper.*, vol. 11, pp. 436-66, 1981.
- [Gent85] W.M. Gentleman, "Using the Harmony operating system," Report of DEE, National Research Council of Canada, NRCC-ERB-966, Ottawa, Ont., Dec. 1983, revised May 1985.
- [Kern84] B.W. Kernighan and R. Pike, *The Unix Programming Environment*, Prentice Hall, 1984.
- [Loo86] R. Loo, "ARIA - A near-real-time graphics package," M.Math Thesis, Univ. of Waterloo, Dept. of Computer Science, 1986.
- [Pleb82] D.A. Plebon and K.S. Booth, "Interactive picture creation systems," Univ. of Waterloo, Dept. of Computer Science, CS-82-46, Dec. 1982.
- [Tann85a] P.P. Tanner and M. Wein, "Parallel input in computer-human interaction," *Proc. 18th Annual Meeting, Human Factors Association of Canada*, Hull, Quebec, Sept. 1985.
- [Tann85b] P.P. Tanner, M. Wein, W.M. Gentleman, S.A. MacKay, and D.A. Stewart, "The user interface of Adagio, a robotics multitasking multiprocessor workstation," *Proc. 1st International Conference and Exhibition on Computer Workstations*, San Jose, Calif., 1985.
- [Tann86] P.P. Tanner, S.A. MacKay, D.A. Stewart, and M. Wein, "A multitasking switchboard approach to user interface management," to appear in *Computer Graphics*, vol. 20, no. 3, August 1986.
- [Tilb76] D.M. Tilbrook, "A newspaper pagination system," M.Sc. Thesis, Univ. of Toronto, Dept. of Computer Science, 1976.

EXPLOITING CLASSES IN MODELING AND DISPLAY SOFTWARE

Turner Whitted and Eric Grant

Department of Computer Science
The University of North Carolina
Chapel Hill, North Carolina

Abstract

The class concept is one component of object-oriented programming systems which has proven useful in organizing complex software. In experimenting with the use of classes for geometric modeling applications, we have devised a class hierarchy that yields some conceptual order in the midst of diverse representations of shapes. Rather than searching for a uniform primitive representation, we accept the diversity and build a framework in which dissimilar models are combined in an orderly manner.

KEYWORDS: geometric modeling, procedure models, object-oriented programming

Introduction

Geometric modeling systems can become extremely complex when design applications demand flexibility in the representation of shapes. A major challenge for programmers who create such systems is to preserve order in spite of this complexity. Trends in this direction include moves toward uniform data representation and very general mathematical representations of shapes. However, the advent of specialized procedural modeling techniques is a step away from uniformity which strains programmers' abilities to cope with the diversity that it presents. We feel that using an object-oriented programming methodology helps to solve this problem.

In spite of the recent interest in object-oriented programming, we have seen only a few published examples of 3-D graphics systems built in an object-oriented environment [Hedelman, Lorensen]. For the most part, the examples that we have seen emphasize the message passing aspects of Smalltalk-like languages and do not pay much attention to the effective specification of classes.

In this paper we describe the class hierarchy of an experimental modeling and display system which we have assembled in order to study the problems of constructing extremely complex geometric objects. We emphasize identifying common elements of geometric procedures which can

be shared among representations. Our guiding principle is that methods should be shared by as many classes as possible and that they should belong to classes as high up in the class hierarchy as possible. In a following section we describe our attempt to define a class hierarchy that meets this criterion.

Diversity of Geometric Representations

Geometric models used in computer graphics have become so diverse that they don't seem to fit any rational scheme of classification. There has been a substantial amount of development of modelers that manipulate polygonal meshes, models that produce parametric surfaces or algebraic surfaces, and unified modeling systems that handle all three representations. Some of these have gone so far as to devise a common representation to ease the difficulty of storing and manipulating the diverse representations.

The widespread use of procedure models [Newell] has complicated the issue even further since the procedures are often restricted to such narrow purposes as creating trees [Bloomenthal], terrain [Fourmier], or grass [Reeves]. To be sure, there are general purpose procedures such as sweeping, and there are generalizations such as graftals [Smith] which provide a common framework for a variety of individual geometric procedures.

The common simplification of reducing all complex shapes to polygonal approximations is no longer feasible when the number of primitive elements exceeds a few tens of thousands. Modification of such complex collections by users is nearly impossible. Common operations such as interference checking and display are confronted with massive amounts of data in this case.

Classes

One of the more successful mechanisms for coping with the complexity of programming is the use of classes. Originally a feature of the Simula language, classes are a central feature of Smalltalk [Goldberg]. Their value is more apparent when one considers that mature languages such as Lisp [Cannon] and C [Stroustrup] have been extended to include classes.

Classes are defined by a set of instance variable declarations and a collection of methods [Robson]. Objects are instances of a class. In a procedural geometric model, the model itself is an object.

The object-oriented organization gives a programmer the benefits of encapsulation and inheritance. To us, the more important property is the inheritance of methods and instance variable declarations. This means that objects that differ only slightly can be cast as members of distinct subclasses of the same superclass. Shared methods and instance variable declarations belong to the superclass. This code sharing has the beneficial side effect of reducing the overall code size. More importantly, development effort is reduced since programmers don't spend time writing the same methods over and over.

Evolution of the Intelligent Modeler

The overall goal of our research is to create detailed geometric models from sparse non-geometric inputs. We share this goal with several similar projects [Feiner, Holynski, Friedell]. Our original intent was to produce a more or less conventional interactive modeling program with an geometric knowledge base as its central element. In operation, the modeler would interpret guidelines provided by the user to invoke rules in the knowledge base which would in turn generate geometric primitives.

This approach was abandoned when we recognized it as an immense, monolithic procedure model. We then settled on a divide and conquer approach to reduce individual components of the modeler to manageable proportions. This then raised the additional problem of coordinating the actions of a multitude of autonomous procedures. For recursive procedures, such as subdivision, which maintain a geometric hierarchy as a product of their operation, we include methods by which mutually constrained objects agree on how each affects the other [Amburn]. So far we have no methods to satisfy mutual constraints for non-recursive procedure models.

Reducing the size of individual intelligent procedure models does not diminish the overall complexity of the modeler. For this we need the class hierarchy described in the following section.

Classes for a Modeling and Display Package

We have built (and continue to expand) an experimental modeling and display package based largely on generic procedure models. A description of the display system is included in this discussion because modeling operations play such an important role in its operation. Figure 1 shows the class hierarchy of the modeling portion of the package. While it may seem logical to organize the classes based on similar properties (i.e. one superclass for curved surfaces, another for polyhedra, etc.), our organization is based on common methods.

Diverse geometric representations can lead to a broad, flat organization of classes. However, our subtree for pro-

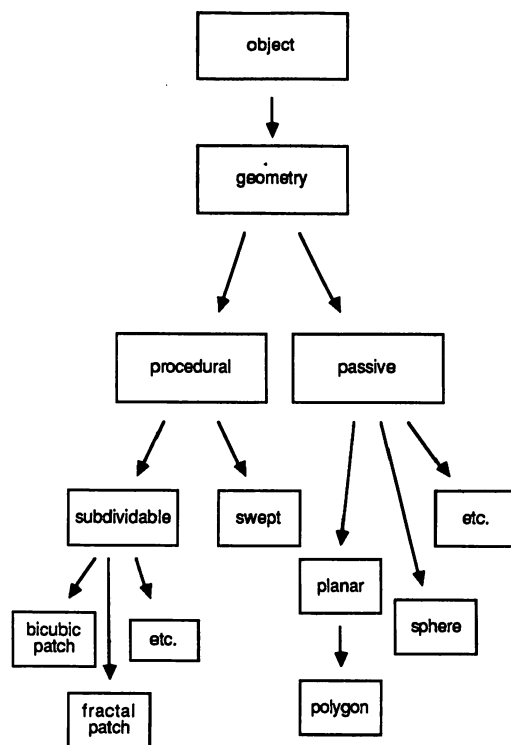


Figure 1. Geometry class hierarchy.

cedural geometry is five levels deep. This is achieved by factoring methods and placing the generic component as high in the tree as possible. Figure 2 is a condensed listing of four levels of one subtree (the *Object* class at the root of the entire class tree is omitted). At the lowest level is the *Bezier* subclass. Only methods specific to the *Bezier* subclass actually belong to it.

To illustrate how methods can effectively be shared, consider the class of subdividable surfaces (the lower three levels of figure 2). A typical display algorithm for this type of surface calls for recursive subdivision to a predetermined level of detail followed by the tiling of polygons formed from the mesh of points generated by the last subdivision stage (figure 3). The subdivision method belongs to the individual subclass. The tiling method, on the other hand, belongs to the class of subdividable surfaces instead of the individual subclasses. Likewise the test for level of detail belongs to the superclass rather than its descendants. This method inheritance is possible because the result of subdivision is a collection of vertices and neither the tiler nor the level of detail test care how the vertices were produced.

The display classes outlined in figure 4 are quite different from ones proposed early in the project. The initial hierarchy had fewer levels and far more classes than the final design. For example, we originally proposed a separate

```
// The class of all geometric elements
class Geometry
{
    // GEOMETRIC INFORMATION

    int numVerts;
    vertex **v;
    BOOL bBoxSet; // boolean for whether bBox is set
    boundingBox bBox; // bounding box parameters

    matrix tform; // transformation matrix

    // SURFACE PROPERTIES

    properties *color; // surface color information

    // RENDERING INFORMATION

    rendParam rendInfo; // rendering parameters (shading)
    float detail; // level of detail

    // note: actual position information is defined by subclasses.
    // 'v' should be set up to point to position information.
    // This allows common methods to be implemented more easily.

public:

    // GEOMETRIC INFORMATION
    boundingBox getBBox(); // return bounding box info
    normal getNormal(); // return surface normal

    // TRANSFORM GEOMETRY
    void rotate(axis, float);
    void translate(float, float, float);
    void scale(float, float, float);
    void transform(matrix); // transform according to supplied mat;

    void tile(); // generate rendering primitives

    void expand(); // expand geometry if possible
    // for example: subdivide

    void collapse(); // reduce geometric complexity

    void textDumpGeometry(); // show geometric info in various forms
    void graphDumpGeometry();
};

// The class of procedurally modeled geometry
class ProcedureGeometry: public Geometry
{
};

// The class of all subdividable procedure models
class SubGeometry: public ProcedureGeometry
{
    int level; // number of times subdivided
public:
    void subdivide(float);
    BOOL terminationTest(float); // level of detail test
    void changeBasis(int);
    void tile();
};

// The class of subdividable bicubic Bezier patches
class Bezier: public SubGeometry
{
    Bezier *parent; // pointer to parent of this patch
    vertex cpoints[4][4]; // the control points
    float umin; // u parameter range in original patch
    float umax;
    float vmin; // v parameter range in original patch
    float vmax;
    Bezier *children[4]; // pointers to subpatches of this patch
public:
    void subdivide(float); // redefine superclass methods
};
```

Figure 2. Geometry class definitions in the C++ language with sample subclasses.

```
// expand: a generic subdivision method
// class: SubGeometry

SubGeometry.expand()
{
    if (this->terminationTest(this->detail) == DONE)
        this->tile();
    else
        this->expandIt();
}

// expandIt: geometry specific subdivision
// class: Bezier

Bezier.expandIt()
{
    // split current patch into four
    // subpatches and expand each
    // one in turn

    new patch1 ; patch1.f00(this) ;
    patch1.expand() ; delete patch1 ;

    new patch2 ; patch2.f01(this) ;
    patch2.expand() ; delete patch2 ;

    new patch3 ; patch3.f10(this) ;
    patch3.expand() ; delete patch3 ;

    new patch4 ; patch4.f11(this) ;
    patch4.expand() ; delete patch4 ;
}
```

Figure 3. A subdivision procedure divided into a generic method and a subclass specific method.

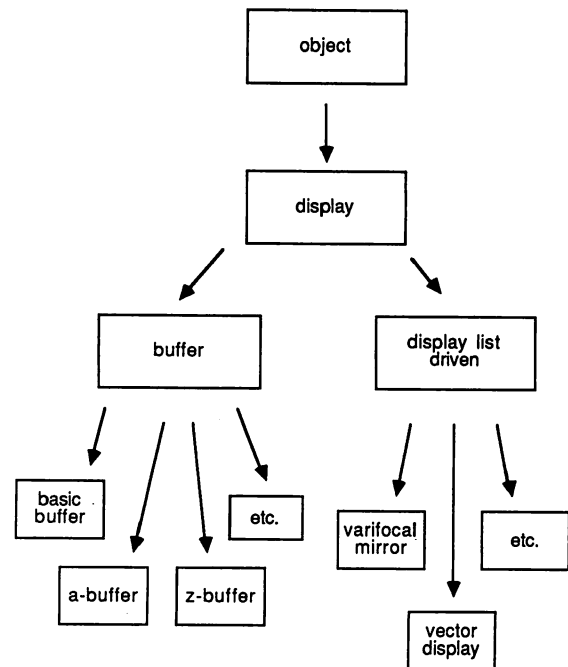


Figure 4. Display classes.

display class for BSP trees [Fuchs]. However, the BSP tree display algorithm is really a sorting method for geometric objects followed by a tiling method which deposits polygons in an image buffer. After factoring the algorithm into its constituent elements, and defining each element as a method of the most appropriate class, we can easily use the BSP tree sorting method in ray tracing to reduce the number of ray-surface intersection tests.

There are a number of commonly used algorithms, such as ray tracing, which are being ripped apart and reconstituted as methods belonging to various geometry and display classes. In general, as we recognize the common elements in different classes, the class tree becomes deeper and narrower.

Conclusion

This paper is not a sermon on the virtues of object-oriented programming. It is concerned instead with its application to modeling and display problems.

Organizing modeling representations into a class hierarchy has drastically altered the way we view geometric models. In particular, we have factored methods into generic and specific parts so that the generic parts can be shared. We have also found ourselves looking for ways to apply those methods that we have on hand to a broader range of geometric problems. We feel that our approach not only makes the programming of the modeling and display package more manageable, but it expands the range of features that the package can support.

Acknowledgement

Our colleague Phil Amburn is responsible for much of the work reported here. We thank Tim Rentsch for his review of the draft. This research is supported in part by Schlumberger-Doll Research.

References

- [Amburn]
Amburn, Phil, Eric Grant, and Turner Whitted, "Managing Geometric Complexity with Enhanced Procedural Models," to appear in *Proceeding of Siggraph '86*.
- [Bloomenthal]
Bloomenthal, Jules, "Modeling the Mighty Maple," *Computer Graphics*, vol. 19, No. 3 July 1985 pp. 305-311.
- [Cannon]
Cannon, Howard, "Lisp with Flavors," Symbolics Inc., internal report.
- [Carlson]
Carlson, Wayne, "An Advanced Data Generation System for Use in Complex Object Synthesis for Computer Display," *Proceedings of Graphics Interface '82*, May 1982, pp. 197-204.
- [Feiner]
Feiner, Steven, "APEX: An Experiment in the Automated Creation of Pictorial Explanations," *IEEE Computer Graphics and Applications*, vol. 5, no. 11, November 1985, pp. 29-37.
- [Fournier]
Fournier, Alain, Don Fussell, and Loren C. Carpenter, "Computer Rendering of Stochastic Models," *Communications of the ACM*, vol. 25, No. 6 June 1982 pp. 371-384.
- [Friedell]
Friedell, Mark, "Automatic Synthesis of Graphical Object Descriptions," *Computer Graphics*, vol. 18, no. 3, July 1984, pp. 53-62.
- [Fuchs]
Fuchs, Henry, Gregory D. Abram, and Eric D. Grant, "Near Real-Time Shaded Display of Rigid Objects," *Computer Graphics*, vol. 17, No. 3 July 1983 pp. 65-72.
- [Goldberg]
Goldberg, Adele, and David Robson, *Smalltalk-80, The Language and its Implementation*, Addison-Wesley, 1983.
- [Hedelman]
Hedelman, Harold, "A Data Flow Approach to Procedural Modeling," *IEEE Computer Graphics and Applications*, vol. 3, no. 9, December 1983, pp. 18-25.
- [Holynski]
Holynski, M., "Meaning Oriented Imaging for Graphics Interface," *Proceeding of Graphics Interface '86*.
- [Lorenson]
Lorenson, William, "An Object Oriented Design of a Graphics Animation System," General Electric CR&D, Internal Report, 1984.
- [Newell]
Newell, Martin, "The Utilization of Procedure Models in Digital Image Synthesis," PhD dissertation, Department of Computer Science, University of Utah, 1975.
- [Reeves]
Reeves, William T., and Ricki Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems," *Computer Graphics*, vol. 19, No. 3 July 1985 pp. 313-322.
- [Robson]
Robson, David, "Object-Oriented Software Systems," *Byte*, vol. 6, no. 8, August 1981, pp. 74-86.
- [Smith]
Smith, Alvy Ray, "Plants, Fractals, and Formal Languages," *Computer Graphics*, vol. 18, no. 3, July 1984, pp. 1-10.
- [Stroustrup]
Stroustrup, Bjarne, *The C++ Programming Language*, Addison-Wesley, 1986.

APPLICATIONS OF WORLD PROJECTIONS

Ned Greene
Computer Graphics Lab
New York Institute of Technology
Old Westbury, NY 11568

Abstract

Various techniques have been developed which employ projections of the world as seen from a particular viewpoint. [Blinn and Newell] introduced reflection mapping for simulating mirror reflections on curved surfaces and their method can be extended to simulate refraction. [Miller and Hoffman] have presented a general illumination model based on world projections. [Greene] has used projections of the world to model distant objects, and [Greene and Heckbert] have used world projections to produce pictures with the fisheye distortion required for Omnimax[®] frames. World projections can also be used as a backdrop for ray tracing or beam tracing.

This paper proposes a uniform framework for representing and utilizing world projections and argues that the best general purpose representation is the projection onto a cube. Surface shading and texture filtering issues related to environment mapping are discussed including approximate methods for obtaining diffuse and specular shading values from prefiltered environment maps. It is noted that obtaining accurate diffuse reflection and antialiasing specular reflection, which are both problematical with ray tracing, can be effectively handled by environment mapping.

Keywords: Environment mapping, reflection mapping, surface shading, texture mapping, cube projection, Mercator projection.

1. INTRODUCTION

Reflection mapping, introduced by Blinn and Newell in 1976, is a shading technique that uses a projection of the world (a "reflection map") as seen from a particular viewpoint (the "world center") to make rendered surfaces appear to be reflecting their environment. The mirror reflection of the environment at a surface point is taken to be the point in the world projection corresponding to the direction of a ray from the eye as reflected by the surface. Consequently, reflections are geometrically accurate only if the surface point is at the world center or if the reflected object is greatly distant. The geometric distortion of reflections increases as the distance from the surface point to the world center increases and as the distance from the reflected object to the world center decreases. To apply reflection mapping to a particular object, the most satisfactory results are usually obtained by centering the world projection at the object center.

This method for approximating reflections can be extended to encompass refraction. Obtaining accurate results, however, requires much more computation since the ray from the eye should be "ray traced" through the refractive object, and in this process the ray usually splits into reflected and refracted components at surface intersections. As with reflections, results are only approximate for geometric reasons. (Note: Simply bending the ray at the surface point and using this as the direction of the refracted ray is not accurate, but may convey the impression of refraction.)

As Miller and Hoffman have described, the concept of reflection mapping may be thought of in more general terms as an illumination model. Essentially, they treat a world projection as an area light source which produces sharp reflections in smooth glossy objects and diffuse reflections in low gloss objects. This is a good model of illumination in the real world, although shadows are not explicitly handled and, as with reflection mapping, results are only approximate for geometric reasons. In order to speak generically of this approach and conventional reflection mapping, the term "environment mapping" will be applied in this paper to techniques for

Omnimax is a registered trademark of Imax Corporation, Toronto, Canada.

shading and texturing surfaces which employ a world projection.

2. STRENGTHS OF ENVIRONMENT MAPPING

Environment mapping is often thought of as a poor man's ray tracing, a way of obtaining approximate reflection effects at a fraction of the computational expense. While ray tracing is unquestionably the more versatile and comprehensive technique, handling shadows and multiple levels of reflection and refraction [Whitted], it is interesting to note that environment mapping is superior in some ways, quite aside from the enormous advantage in speed. Obtaining accurate diffuse reflection and antialiasing specular reflection are both problematical with ray tracing since it point samples the three dimensional environment (Amanatides' approach of ray tracing with cones is an exception). Environment mapping can effectively handle these problems by filtering regions of the world projection. Under many circumstances, for example when a large area light source illuminates a low gloss object, the subjective quality of reality cues produced by environment mapping are superior to those produced by unadorned ray tracing. While it is noted that refinements to ray tracing proposed by [Cook-Porter-Carpenter] and [Amanatides] address the problems of aliasing and diffuse reflection, their methods increase several-fold the already extreme computational cost. (Incidentally, there is an interesting parallel between the way these refinements work and the use of texture filtering in environment mapping: both attempt to integrate over a region of the environment.) It should also be mentioned that ray tracing and environment mapping can be used in combination where foreground objects are ray traced and a world projection representing distant objects serves as a backdrop (Lucasfilm's "1984" serves as an example [Cook-Porter-Carpenter, Figure 8]).

3. AN EXAMPLE

The dimetrodon lizard of Plate 1 was rendered with reflection mapping (i.e. environment mapping with mirror reflections). Inspection of this image reveals an inherent limitation with reflection mapping: since the reflecting object is not normally in the world projection the reflecting object cannot reflect parts of itself, e.g. the legs are not reflected in the body. (Actually, this limitation can be partially overcome by using different world projections for different parts of an object.) But overall, reflection mapping performs well in this scene. The horizon and sky, which are the most prominent reflected features, are accurately reflected due to their distance from the reflecting object. The reflections of the tree and the foreground terrain are less accurate because of their close proximity, but surface curvature makes this difficult to recognize. As this example shows, reflections don't need to be accurate to look realistic, although attention should be paid to scene composition. Planar reflecting surfaces may cause problems, for example,

since the distortion of reflections in them may be quite noticeable.

4. RENDERING A CUBE PROJECTION

Use of environment mapping presupposes the ability to obtain a projection of the complete environment. Suppose that we wish to obtain a world projection of a three dimensional synthetic environment as seen from a particular viewpoint. One method is to position the camera at the viewpoint and project the scene onto a cube by rendering six perspective views, each with a 90 degree view angle and each looking down an axis of a coordinate system with its origin at the viewpoint. The world projection used to shade the lizard in Plate 1 was created in this manner and the resulting cube is shown unfolded in Plate 2. This method of creating world projections has also been used by Miller and Hoffman as an intermediate step in creating a Mercator projection of the world, the format they prefer for environment mapping. Blinn and Newell also use a Mercator projection.

5. SURFACE SHADING AND TEXTURE FILTERING

Light reflected by a surface is assumed to have a diffuse component and a specular component. The diffuse component represents light that is scattered equally in all directions, the specular component represents light that is reflected at or near the mirror direction. This discussion of surface shading will be confined to obtaining diffuse and specular illumination from a world projection; the problem of combining this information along with surface properties (color, glossiness, etc.) to obtain the color reflected at a surface point will not be considered. See [Cook and Torrance] for a general discussion of surface shading, [Miller and Hoffman] for a discussion of shading in the context of environment mapping.

Diffuse illumination at a surface point comes from the hemisphere of "sky" in the world projection centered on the surface normal, and it can be found by filtering the region of the world projection corresponding to this hemisphere. Filtering should be done according to Lambert's law which states that the illumination coming from a point on the hemisphere should be weighted by the cosine of the angle between the direction of that point and the surface normal.

A region of the world projection should also be filtered to obtain the specular illumination component. Figure 1 shows the cone of "sky" reflected by a surface which corresponds to a pixel in the output image. An obvious approach to determining the specular illumination at a pixel is to find the region of the world projection subtended by the corresponding "reflection cone" and then average the pixels in this region. While this approach will produce reasonable results, they will not be optimal for several reasons. One problem is that the size of the region filtered should be influenced by surface roughness

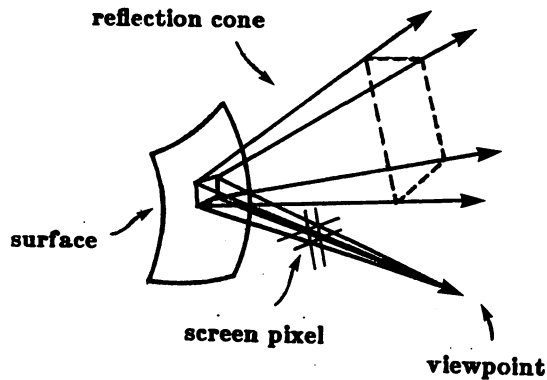


Figure 1. Rays from viewpoint through corners of a screen pixel are reflected by a surface, defining a reflection cone.

at the pixel level, since rough surfaces scatter specular illumination (thus, a larger area should be filtered if the surface is rough). Secondly, for theoretical reasons filtering should not be restricted to the region of texture corresponding to pixel bounds, and averaging of pixel values should be weighted by proximity to the center of the region being filtered. Subsequent references to reflection cones are made bearing in mind that they indicate only the approximate boundaries of regions to be filtered. See [Heckbert] for a general discussion of texture filtering.

While the details of shading formulas are beyond the scope of this paper, a rough rule for shading monochrome surfaces at a surface point is:

$$\text{reflected color} = dc * D + sc * S, \text{ where}$$

dc is the coefficient of diffuse reflection
D is the diffuse illumination
sc is the coefficient of specular reflection
S is the specular illumination

(Note: dc and sc depend mainly on surface glossiness)

Plate 4 shows three monochrome spheres reflecting the environment of Plate 2. The relative weighting of the diffuse and specular components varies from completely diffuse on the left (a Lambertian reflector) to completely specular on the right (a perfect mirror).

6. CUBE PROJECTIONS VS. MERCATOR PROJECTIONS

When world projections are rendered from three dimensional models (rather than being photographed or painted), the cube representation of the world is preferred to a Mercator projection for reasons of computational efficiency and image quality. Rendering the cube projection is normally required in both cases. The further step of creating a Mercator projection from the cube projection requires additional computation, and the added generation of filtering can only degrade image clarity.

Moreover, Mercator projections are non-linear which complicates texture filtering since the region of texture subtended by a reflection cone does not have a polygonal boundary. (With cube projections, reflection cones always subtend polygonal regions of cube faces.) Figure 2 shows the regions subtended by the same reflection cone in Mercator and cube projections. Accurate filtering of the subtended region in the Mercator projection (the upper region) is problematical due to its irregular shape and the fact that pixels in the projection correspond to widely varying areas of "sky." Admittedly, filtering a cube projection presents a different problem: the multiplicity of regions to be filtered, five in the example of Figure 2. Fortunately, these problems occur only when surface curvature is high at the pixel level, which is not usually the case. Low surface curvature produces narrow reflection cones which map to approximately quadrilateral areas in a Mercator projection and which usually map to a quadrilateral on a single face of a cube projection.

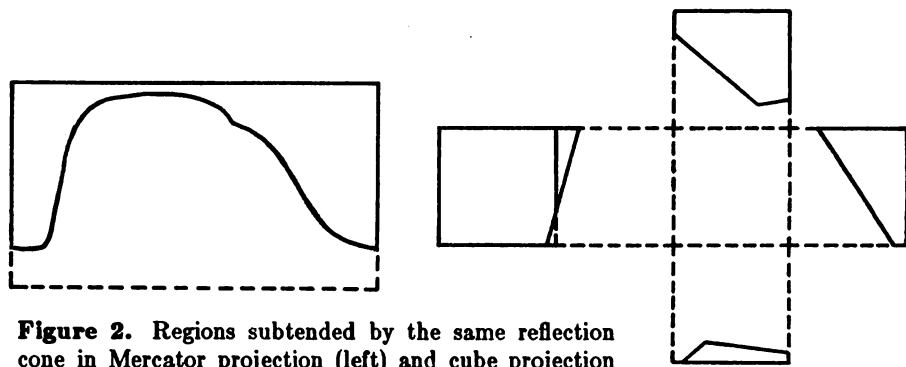


Figure 2. Regions subtended by the same reflection cone in Mercator projection (left) and cube projection (right). The reflection cone covers about 3/8 of the world.

7. PREFILTERED WORLD PROJECTIONS

As noted above, the width of a reflection cone depends on surface curvature so the area of "sky" reflected at a pixel can be arbitrarily large. A sphere covering a single pixel, for example, reflects nearly the entire world. Thus, environment mapping normally requires filtering large areas of the world projection at some pixels, even to produce reflections in mirror surfaces, so approximate filtering methods utilizing prefiltered texture greatly enhance efficiency.

As Miller and Hoffman describe, a diffuse illumination map can be created by convolving the world projection with a Lambert's law cosine function, a kernel covering one hemisphere of the world. This map, which is indexed by surface normal, may be thought of as indicating the colors reflected by a monochrome spherical Lambertian reflector placed at the world center. Since the diffuse illumination map has little high frequency content it may be computed and stored at low resolution and accessed with bilinear interpolation. Thus, prefiltering the world projection in this manner reduces the problem of finding the diffuse illumination at a surface point to a table lookup. Plate 3 is a magnified diffuse illumination map in Mercator projection format corresponding to the world projection of Plate 2. Incidentally, since diffuse illumination maps usually change only subtly from frame to frame, for animation applications it may suffice to create these maps at intervals (say every tenth frame) and interpolate between them.

Specular illumination can be obtained using fast, approximate filtering techniques developed for texture mapping surfaces. Our present implementation of environment mapping uses a prefiltered cube projection in the form of six "mip maps" [Williams], one for each cube face. Mip mapping is fast, but can only filter square regions of texture, so results are only approximate (in the example of Figure 2, the polygonal areas on the cube faces would need to be approximated by squares). [Crow] and [Perlin] have proposed similar approximate filtering techniques.

8. CHEAP CHROME EFFECTS

Environment mapping also has wide application where the objective is to produce a striking visual effect without particular regard for realism. Often the intent is to give objects a chrome plated look and the content of reflections is unimportant. In Robert Abel and Associate's "Sexy Robot" animation, for example, the reflection map was a smooth color gradation from earth colors at low elevations to sky colors at high elevations, and at a given elevation color was constant [Bytes]. For applications of this sort, a colormap (or other one dimensional color table) suffices to specify the color gradation, and rendering computation can be greatly reduced by filtering this table instead of performing two dimensional

texture filtering. Since an environment map isn't used, memory requirements are minimal and no setup time is required to prefilter texture. Highlights can be produced by adding point light sources independently of environment mapping.

9. MODELING BACKGROUND OBJECTS WITH WORLD PROJECTIONS

While the discussion thus far has been confined to using world projections for surface shading, they can also be used to model background objects. As described in [Greene], the sky component in frames of moving camera animation can be modeled as a half-world projection, for example the upper half of a cube. As the camera moves through the scene the appropriate region of the projection comes into view. Of course the advantage of this technique is speed: a scene element (in this case the sky) can be rendered from the world projection with texture mapping which, for complex scenes, is much faster than rendering from corresponding three dimensional models. This method assumes that objects in the sky are greatly distant from the camera, and results are only approximate when this is not the case. Plate 5 is a frame from animation in which the sky component was rendered from texture painted on a half-world projection.

Since half-world models cover the whole sky, they are particularly useful for creating world projections for environment mapping. The sky component of Plate 2 was modeled by projecting a 180 degree fisheye photograph of sky onto a half cube, shown in isolation in Plate 6. This model served two purposes in producing the picture of Plate 1, modeling the sky in the background and making the world projection which was used to shade the lizard. Incidentally, using photographic texture for sky models is an attractive option due to the difficulty of synthesizing complex sky scenes with realistic cloud forms and lighting effects, and using a 180 degree fisheye lens allows the whole sky to be photographed at once, thus avoiding problems associated with photo mosaics.

In scenes of animation where the camera rotates but doesn't change location, this approach to modeling sky can be extended to modeling the whole background. In this case, a single world projection centered at the camera is generated and the background can be rendered directly from this projection for the frames of animation. The geometry of the scene is faithfully reproduced regardless of the distance of objects in the scene from the camera; results are not just approximate. There are also limited applications of this technique to moving camera animation. A world projection of the distant environment centered at a typical camera position can be used to render distant objects in the scene while the near environment is rendered from models at each frame and then composited with the distant environment. This approach is analogous to using foreground and background levels in cel animation. As before, it is convenient to represent the world as a cube projection since

it can be rendered directly from three dimensional models.

Rendering background objects from a world projection is a form of texture mapping since each pixel in the output image is rendered by determining the corresponding region in the world projection and then filtering a neighborhood of this region. Assuming that the world is represented as a cube projection, the cube faces should have substantially higher resolution than the output frames since only a small part of the world is subtended by the viewing pyramid for typical view angles. For example, a viewing pyramid with a 3:4 aspect ratio and a 45 degree vertical view angle subtends only about six percent of the world. Prefiltering the world projection is unnecessary since only small regions of texture are filtered.

This approach to rendering background objects suggests a method for performing motion blur. The region of texture traversed by the projection of an output pixel in the time interval between frames is determined, and then this region is filtered. The simplicity of performing accurate motion blur in this situation is due to the two dimensional nature of the model. The filter employed should be spatially variant because different output pixels may traverse differently shaped paths (this occurs, for example, if the camera rolls). Approximate results can be obtained by filtering elliptical regions in the world projection. Greene and Heckbert present an efficient method of filtering arbitrarily oriented elliptical areas.

10. NON-LINEAR PROJECTIONS

In addition to using world projections to generate perspective views of an environment, they can also be used to create non-linear projections such as the fisheye projection required for Omnimax frames. (The screen in an Omnimax theater is hemispherical and film frames are projected through a fisheye lens [Max].) Greene and Heckbert have obtained Omnimax projections of three dimensional scenes by projecting the scene onto a cube centered at the camera at each frame and then filtering regions of the cube faces to obtain pixels in the output image. This technique is very similar to the method described in the preceding section for rendering background objects from world projections. Plate 7 is an Omnimax projection made from the cube projection of Plate 2.

11. CONCLUSION

The projection of the environment onto a cube is a convenient and efficient format for world projections for the applications cited in this paper. In the context of a graphics system which employs world projections for multiple applications, the advantages of using a standard format are obvious: projection and filtering software is simplified, multiplicity of picture formats is avoided, etc. Generally, the methods described are ways of approxi-

mating a three dimensional problem with a two dimensional problem in order to reduce computational expense: environment mapping approximates ray tracing, rendering background objects from world projections with texture mapping approximates image rendering from three dimensional models. The subjective quality of reality cues in images produced with these approximate methods often compares favorably to results obtained by more expensive image generation techniques, and for complex environments approximate techniques may be the only practical way of producing animation having the desired features with moderate computing resources.

Acknowledgements

My thanks to Paul Heckbert who collaborated to incorporate environment mapping in his polygon renderer, who devised the "elliptical weighted average filter" used to create Omnimax projections, and who provided a critical reading of this paper. Thanks also to John Lewis, Tom Shermer, and John Schlag for their comments on the paper and to Dick Lundin who assembled and rendered the scene of Plate 1. Ariel Shaw and Rich Carter provided photographic assistance.

References

- Amanatides, John, "Ray Tracing with Cones," *Computer Graphics*, vol. 18, no. 3, July 1984, pp. 129-135.
- Blinn, James F., and Martin E. Newell, "Texture and Reflection in Computer Generated Images," *C.A.C.M.*, vol. 19, no. 10, Oct. 1976, pp. 542-547.
- Bytes, Torrey, "Displays on Display: Commercial Demonstrates State-of-the-Art Computer Animation," *IEEE Computer Graphics and Applications*, April 1985, pp. 9-12.
- Cook, Robert L., Thomas Porter, and Loren Carpenter, "Distributed Ray Tracing," *Computer Graphics*, vol. 18, no. 3, July 1984, pp. 137-145.
- Cook, Robert L., and Kenneth E. Torrance, "A Reflectance Model for Computer Graphics," *Computer Graphics*, vol. 15, no. 3, August 1981, pp. 307-316.
- Crow, Franklin C., "Summed-Area Tables for Texture Mapping," *Computer Graphics*, vol. 18, no. 3, July 1984, pp. 207-212.
- Greene, Ned, "A Method of Modeling Sky for Computer Animation," *Proceedings of the First International Conference on Engineering and Computer Graphics*, Beijing, China, Aug. 1984.

Greene, Ned, and Paul S. Heckbert, "Creating Raster Omnimax Images from Multiple Perspective Views using the Elliptical Weighted Average Filter," *IEEE Computer Graphics and Applications*, June 1986(?).

Heckbert, Paul S., "Survey of Texture Mapping," *Proceedings of Graphics Interface '86*, May 1986.

Max, Nelson, "Computer Graphics Distortion for Imax and Omnimax Projection," *Nicograph '83 Proceedings*, Dec. 1983, pp. 137-159.

Miller, Gene S., and C. Robert Hoffman, "Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments," *Siggraph '84 Advanced Computer Graphics Animation seminar notes*, July 1984.

Perlin, Kenneth, "Course Notes," *Siggraph '85 State of the Art in Image Synthesis seminar notes*, July 1985.

Whitted, Turner, "An Improved Illumination Model for Shaded Display," *C.A.C.M.*, vol. 23, no. 6, June 1980, pp. 343-349.

Williams, Lance, "Pyramidal Parametrics," *Computer Graphics*, vol. 17, no. 3, Aug. 1983, pp. 1-11.

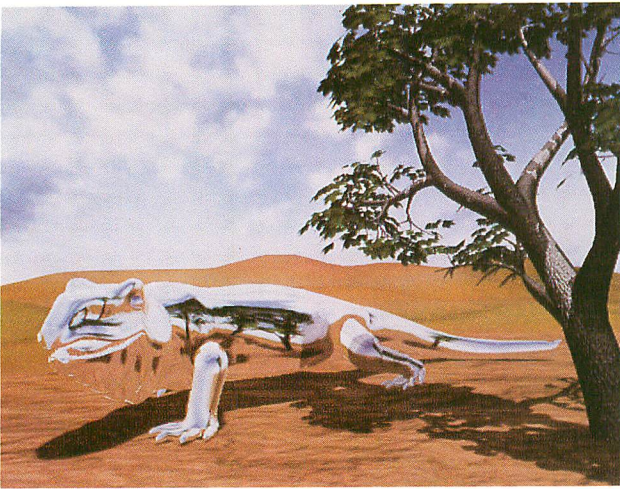


Plate 1. Dimetrodon lizard model reflecting its environment. (Lizard modeled by Dick Lundin, tree modeled by Jules Bloomenthal)

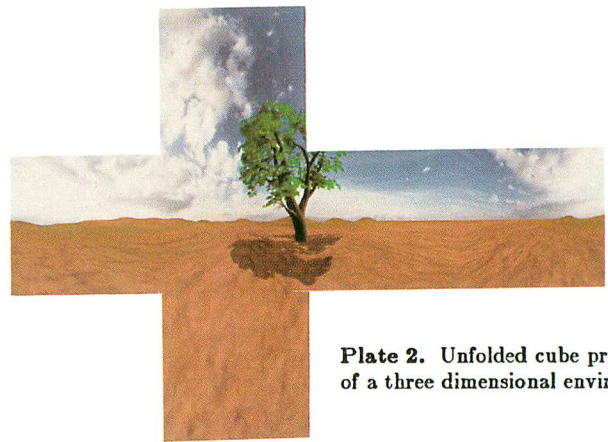


Plate 2. Unfolded cube projection of a three dimensional environment.

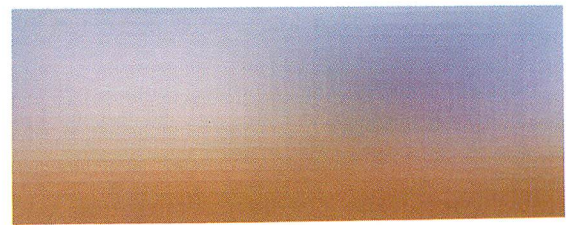


Plate 3. Diffuse illumination map of the environment of Plate 2 (Mercator format).



Plate 5. Frame from animation utilizing a half-world sky model with painted texture. (Note: The reflection in the lake was *not* produced with reflection mapping.)



Plate 4. Spheres reflecting the environment of Plate 2. Lefthand sphere is a Lambertian reflector, righthand sphere is a perfect mirror.

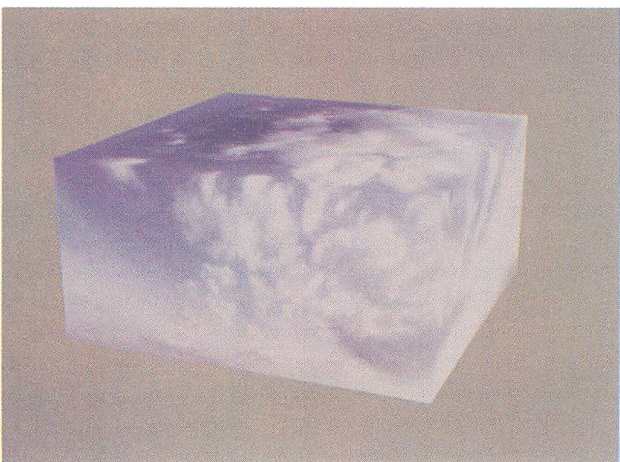


Plate 6. Half-world sky model made with photographic texture.



Plate 7. Omnimax projection of the environment of Plate 2.

ANIMATING HUMAN FIGURES: PERSPECTIVES AND DIRECTIONS

Norman I. Badler
Department of Computer and Information Science
Moore School D2
University of Pennsylvania
Philadelphia, PA 19104

ABSTRACT

The overall goal of our work is human motion understanding. In particular, motion performance, observation, description, and notation impacts the form of a motion representation. A representation can be verified by a computer graphics performance, and thus the effective control of natural-appearing human figure movement is a significant and challenging goal. Characteristics of a computationally realizable human movement representation are discussed, including distinctions between hierarchic levels, kinematics, and dynamics. The qualitative factors of Effort-Shape notation are used to suggest extensions to existing movement representations in directions consistent with known characteristics of human movement and conventional animation. We show how useful and expressive motion qualities may be at least approximated by a combination of kinematics and dynamics computations, with kinetic control modulated by acceleration and decelerations derived from existing interpolation methods. Interactions between motions by phrasing, temporal properties, or relationships may be described and executed within an appropriately detailed model.

RÉSUMÉ

L'objet de notre étude est la compréhension du mouvement humain. Plus particulièrement, le fonctionnement du mouvement, son observation, sa description et sa notation ont un impact sur l'organisation de la représentation du mouvement. Celle-ci peut être contrôlée à l'aide de l'informatique graphique, mais un contrôle adéquat du naturel de l'apparence du mouvement du corps humain est un défi à relever. Différentes caractéristiques de la réalisation du mouvement par informatique sont examinées. On distingue notamment la cinématique, la dynamique et les niveaux hiérarchiques. Les facteurs qualitatifs d'une notation "Effort-Shape" précise sont utilisés pour évoquer l'extension de la représentation actuelle du mouvement vers une direction compatible avec les caractéristiques courantes du mouvement et de l'animation. Nous démontrons comment certaines qualités significatives du mouvement peuvent être approximées par la dynamique et la cinématique avec le contrôle de la cinétique modulée par l'accélération et la décélération ces deux dernières étant dérivées par les méthodes d'interpolation conventionnelles. L'interaction entre l'expression du mouvement et les propriétés temporelles peuvent être décrites et exécutées selon les limites d'un modèle pertinemment détaillé.

KEYWORDS: Human movement, motion understanding, movement representation, computer animation, simulation, computer graphics, dynamics.

EXTENDED ABSTRACT

INTRODUCTION

A significant portion of our activities and perceptions are associated with the performance, observation, description, or recording of human movement. It is a challenge to the current state of knowledge in Computer Science to similarly represent, simulate, and integrate these differing manifestations of human movement since they touch on such seemingly diverse areas as computer graphics, computer vision, robotics, and computational linguistics [6]. In this exposition we shall discuss the philosophy and methodology behind our research into the computational understanding of human movement, concentrating on the issues of movement representation, movement synthesis, and task specification. While our primary emphasis will be on performance, that is, the animation or simulation of natural human motion, we cannot avoid inquiring what our representational decisions would imply for a general theory of human movement understanding.

We will try to examine human movement in the most global view possible, namely, that a movement representation should be at least sympathetic to the needs and character of each modality: performance (or control), observation, language description, or symbolic recording. Our own research, and certainly that of others, has touched all these areas: for example, computer graphics for human motion synthesis [9, 16, 65, 33, 41, 38, 21], computer vision for motion and shape analysis [46, 1, 36], movement notations for symbolic motion representation [29, 63, 9, 15], language analysis for motion verb characterization [45, 4, 23], and robotics for path planning and goal-directed behavior [35, 34]. Having originally examined motion descriptions based on visually-observable data [4], the inadequacy of this view by itself is keenly felt. Such descriptions may serve as a target for information reduction, but are apt to be the product of convenience dictated by the observational task at hand. Such a description differentiates between phenomena of interest, possibly incorporating rudimentary notions of direction, velocity, and shape. Likewise, representations derived solely from language [56] omit essential information needed to reconstruct an acceptable performance.

By turning to representations derived from graphical performance or physical object control (for example, robotics), we get a different perspective. In particular, representations based on these end products will have the property that a graphical or physical performance will verify that a representation is adequate to characterize some (hopefully broad) class of human movement. It is

this *adequacy* that permits experimentation based on empirical data (say from observed movements) and parametric variation to control or tune the result.

The role of natural language descriptions is to expose the salient features of human motion interpreted (by a culture) as significant events. In particular, we find language has evolved rich verb and adverbial vocabularies to permit the description and expression of subtle movements. In fact, language goes even further by imputing behavior, emotion, and intent to movement, even when that motion is not obviously attributable to human-appearing agents [44]. While such information is available subconsciously via our cognitive systems, it may also be instantiated in language (or physically acted out, for that matter). Therefore we assume the existence of a transformation which maps some of these subconscious perceptions into tangible (and essential) components of a motion representation. It appears that some of this information can be captured; how much is not clear, though we will propose a model here.

Finally, we use movement notations as a source of symbolic representations derived from empirical observation and analysis over many years by many observers of numerous subjects. The impact of such systems is that they provide one of the only possible bases for establishing *completeness*: that is, does the representation cover, in its variations, the known scope and range of human movement? Language also provides some of this scope, but does not lend itself so readily to analysis.

We proceed by examining some of the representational issues which arise in considering the influence of these requirements.

REPRESENTATION REQUIREMENTS

Movements of human or robot agents may be characterized at many different levels. A purely geometric level of description as changing coordinates, though necessary, is insufficient as a comprehensive basis for understanding motion. A simple gesture such as closing the hand may be described by joint angles, by paths of the fingertips, by flexion of muscles, by the concept "grasp," or by the intention "shake hands." Each type of description is useful in different contexts, and a natural hierarchy of levels seems to appear. To discuss a movement representation therefore is to establish what descriptive levels are important and what attributes or characteristics are adequate to completely "cover" the space of possible movements at each level. We will return to this issue later, after establishing a plausible representation scheme in which to formulate higher level motion or action descriptions [22].

Viewing movement hierarchically focuses attention on descriptive or conceptual levels, that is, the refinement or generalization of a movement at a different level of detail. Performance of a particular motion, however, requires the interaction or combination of effects from many sources. While geometric object descriptions lend themselves to a hierarchic view [18, 8, 42], motions are dictated by simultaneous interacting influences. Muscle tension, external forces, joint limits, path constraints, expressive purpose, intention, and the context of temporally adjacent activities all affect human movement. A more general approach to movement understanding therefore would cover at least the following aspects of a motion.

- The geometry, kinematics, and dynamics of the agent.

The individual differences in people and their anthropometry

must be taken into account. Motion is significantly affected by the kinematics of jointed objects, such as joint limits, reachable points, and comfort zones. Dynamics describes the force or effort influencing motion, whether actual or perceived, and may be independent of motion path. Dynamics also involves the inherent strength of the agent to initiate or resist motion.

- Any goal-directed or intentional acts the movement was part of.

Much human motion is intentional, even if unconscious: the achievement of reach goals, negotiation through a space, maintenance of balance, and comfortable distribution of weight.

- The agent's attitude toward the environment, and its general mode of behavior.

The interpretation of any particular motion is highly dependent on the environmental and personal context: thus a "threatening" gesture in a social context may be merely "defensive" in an athletic one. Motions which are part of an ongoing task or activity may be perceived as more global entities rather than isolated movements.

- What, if anything, it signified.

For example, sign language research [31] shows that certain seeming variations in a movement are understood as the same sign, while others are not. Often movements along the same spatial path and toward the same spatial goal may signify very different intents, such as "touch," "press" and "punch."

- Any synchronization or concurrency relations the movement depends on or is derived from.

Motions may occur in isolation, in sequence, in parallel, or in any overlapped or superimposed combination. Some of these relationships were studied in the motion context [10], in language [2, 62], and in task-level reasoning [61, 22]. They may also overlap, mask, dominate, accentuate, or modify one another, as has been demonstrated with facial motions [52, 53]. The movements may occur compressed or extended in time, or be subject to environmental constraints or control requirements. For example, the actions of a group of athletes on a team is subject to the rules of the game as much as the particular instantaneous circumstances of the play.

Of course, these factors are not orthogonal to one another, but interact and interrelate in complex ways. Part of our task is therefore to organize motion information so that we can hope to control motion to the extent that the different factors can be investigated at appropriate levels.

The central "core" of the movement understanding methodology is a movement representation and its interpretation by computer simulation. The reason we insist upon interpretation will be clarified further in the next section. In succeeding sections we will examine particular aspects of the motion representation and show how each component is essential to effective motion synthesis and how its semantics might be implemented.

MOVEMENT REPRESENTATION

In keeping with the general concerns expressed above, we enunciate several criteria deemed essential to the design of an effective motion representation. To focus the effort, we will define a movement representation as a system in which *any movement may be decomposed into "primitives" with implementable semantics*. We require these primitives to meet certain constraints:

- descriptive significance

This issue implies that mere visual images are not sufficient for a motion representation; even an extensive "film library" is not in the form of primitives that may be readily used as the basis for simulating arbitrary motion patterns. There is no index upon which similarity or differences between two motions may be easily judged. There may not even be agreement between observers as to the name or type of motion being performed. The fact that most imagery is two-dimensional is an additional complication, but if the images were from multiple viewpoints or even holographic, the objection would still stand.

A similar objection can be raised to descriptions consisting of natural language text. Though there may be cultural agreement on the meaning of an utterance, the actual process of converting the description to action may be subject to widely varying interpretations, for example, via "acting."

- modifiability through generally accessible methods

This issue implies that a motion representation must permit the symbolic or computational modification of a motion primitive in order to create a wide class of related or similar motions. "Generally accessible" implies eliminating choices such as libraries of artist-drawn animations, since the creation of natural-appearing hand-drawn animations is not a widespread skill. At the minimum, this constraint argues for parametric descriptions, though we need not commit to a specific set of parameters yet.

- independence of specific individuals

This issue again rejects the film or artist-drawn library approach, and also disallows more detailed but still joint- or segment-specific motion data collected from an individual. Thus while such motion may be used as the basis of a specific animation [15, 24, 21], it is not obvious how such a motion would change if it were applied to another individual with different body dimensions, weight, strength, posture, etc.

- independence of specific motion characteristics

This issue emphasizes the need for a parametric approach, though now the problem is the motions within an individual and the possible ways they can be combined, compounded, executed in parallel or sequence, inhibit or permit other motions, etc. Thus the primitives must describe possible actions of body components and be subject to synchronization and modification by other primitives. In addition, we expect physical factors to be separable: for example, the path of a motion should be separable from the kinetics of motion along the path. Again, representations of the library type cannot deal effectively with the computational explosion of possibilities inherent in arbitrary human motion.

In constructing a movement representation we have been very concerned with its capabilities to describe sufficient information for a "performance" by computer synthesized graphic images [9]. This point of view has been very fruitful in deciding what characteristics of a movement description and hence of an adequate representation, are necessary. The important concept is that movement synthesis considerations demand consistent implementable semantics. If a computer system could produce any movement specified by the appropriate descriptive parameters, then it would also verify that a representation was an adequate knowledge base with which to describe or notate observed movement. Thus, for example, if the representation cannot express the differences between "press" and "punch," it would not have sufficient means to distinguish these actions if actually observed.

Symbolic representations of many movement properties are found in Labanotation [29], a movement notation system originated

over 50 years ago by Rudolf Laban. Though several notation systems exist, few come close to meeting the criteria for a movement representation. We initially studied Labanotation [9], basing the choice on several factors deemed essential for effective motion specification:

- its redundant means of expressing a movement
- its methods for handling sequence, concurrency, and phrasing
- its capabilities for arbitrary frames of reference
- its incorporation of goal-directed actions
- its essentially "digital" symbol system.

We abstracted these Labanotation properties into a set of five "primitive movement concepts" [63] (*directions*, *revolutions*, *facings*, *shapes*, and *contacts*) concerned only with the location and relations of body joints or surfaces in space. Significantly, these primitives do not cover dynamic effects (force, acceleration, torque, etc.), muscular movements (bulges, contractions, etc.) or facial expressions [52, 48]. Thus a motion specification in this system actually describes the final goal and some constraints on the path rather than the internal method by which it is achieved [5]. *Directions* generally describe positions to be achieved by body parts, or directions in which the entire body is to move. *Revolutions* include rotations and twists by given angles. *Facings* are goal-directed rotations which require a body surface to achieve a desired orientation. A *shape* is either a path along which a body or body part moves, or a spatial shape (position or configuration) which some subset of the body is to achieve. *Contacts* are generally relationships such as touches, supports, contains, etc., between two or more bodies, body parts, or environmental points. All the primitives share notions of duration, fixed end, and reference coordinate system.

We have recently come to view movement somewhat differently. The evolution of this early motion representation is motivated not only by current efforts in three-dimensional computer animation [38, 41], but also by practice in robotics [50, 40, 27, 49, 20] and motion analysis [46, 60]. We distinguish four different kinds of movement primitives:

- "Changes": rotations by a given angle or translation along a given path or direction
- "Goals": achievement of a given location and/or orientation for a body point [35]
- "Paths": curves in space along which points may move
- "Dynamics": kinetics or forces which control or affect a motion

The former "primitive movement concepts" are easily subsumed into the first three of these four primitives. The new primitive, *dynamics*, will be discussed in the next section. A comparison of the categories of the "old" representation [9, 63] with respect to this new representation appears in Table 1.

In Table 1, a *reach* refers to the kinematic achievement of a location in space by some body point and an *orientation* to the kinematic achievement of an orientation of a body point. The "key-parameter" concept refers to a set of parametric values for particular manipulable variables of the body such as joint angles, reach position, body location, etc.

Changes, goals, and paths must have associated with them durations, starting times, and reference coordinate systems. We can assume that the original specification is adequate in that regard [63]. Items such as fixed ends of a reach goal are indicated by zero changes in that body point in an appropriate coordinate reference

Table 1: Comparison of "old" and "new" movement representations.

"old"	"new"
DIRECTION (movement)	Change in position
DIRECTION (position)	Reach goal
REVOLUTION (rotate)	Change in orientation
REVOLUTION (twist)	Change in orientation
FACING	Orientation goal
SHAPE (movement)	Sequence of reach goals or "key-parameter" locations
SHAPE (position)	"Key-parameter" positions
CONTACT	Sequence or set of reach and orientation goals

frame [5, 25]. Thus the shoulder might be the fixed end for an arm reach to position and orient a hand with respect to some object. The former *contact* primitive is subsumed into time-marked sets of one or more goals achieved sequentially and in parallel as needed. The semantics of *determining* those goals is left to a higher level process [7, 65, 23, 22].

The task of synchronizing concurrent actions and handling multiple constraints is passed to a control system rather than being explicitly embedded in the representation. A parallel control algorithm had been advocated earlier for this purpose [9]. The essential features of this control were

- joint "processors" which interpreted parallel streams of motion primitive "instructions" as programs,
- a special processor to handle movements of the center of gravity, and
- a global monitor to synchronize local changes to a global, constrained, body model and thus process concurrent overlapping motion primitives.

We can relax the control model by viewing the body parametrically, that is, any specified point on the body may be controlled by specifying a sequence of one or more values over time for it. Paths are themselves a sequence of parameter values. The parameter values may be affected by more than one primitive, for example, the position of the body's center of gravity may be affected by the path of movement, inertia, and external forces [9, 25]. It is the responsibility of the animator and the simulation semantics to resolve any discrepancies [10, 53]. The particular interactions of the *dynamics* primitives are new and will be examined carefully in the next section.

DYNAMICS

A key feature of human movement virtually ignored in earlier representation efforts is its dynamic quality, that is, the manner in which the body moves in terms of *force*, *effort*, *exertion*, *energy*, etc. This may be more significant, in an expressive or intentional sense, than the actual path. For example, variations in dynamics can alter

the message conveyed in American Sign Language [31, 39]. Dynamics considerations appear only implicitly in the representations derived from the study of movement notation systems because:

- Labanotation (or for that matter, nearly any other notation) does not convey dynamic information other than timing (duration) and perhaps *accent*,
- Motion semantics have been mostly concerned with *visually smooth* implementation of each primitive motion, not of the *details* of that motion during its execution nor with its continuity in the context of temporally adjacent motions, and
- The computational models must include capabilities for understanding some minimal physics associated with body mass, force, inertia, gravity, balance, etc. [10].

Computer animation done without concern for motion dynamics looks flat or mechanical at best; discontinuous or jerky at worst.

Previous efforts at incorporating dynamics into computer generated animation have focused on explicit velocity or acceleration functions [43, 58, 17, 26], artist-drawn keyframes [14, 54], smooth spline functions [57, 59, 32], or actual human dynamics [15, 11, 66, 24]. The problem has been investigated more mathematically in mechanics [47, 30, 51], biomechanics [55], and robotics [27, 37, 13, 28], though the latter has been much more concerned with computational efficiency. Recently, such techniques have been applied to human or articulated figure dynamics [3, 64, 25]. Our own examination of the dynamics problem has focused on alternative notation systems combined with physical and graphical motion models suited to the complexity of the human figure.

In searching for a representational basis for the dynamic qualities of movement, we examined a notation system complementary to Labanotation called Effort-Shape notation [19, 12]. Unfortunately, the semantics of this system are not defined quantitatively, so we have interpreted it freely to produce something more amenable to computation. We believe this to be a reasonable approach since our intent is not to "computerize" Effort-Shape, or another notational system as we and others have attempted

to do. Rather, we use these systems to aid in comprehending the scope and variety of human movement so that our representations are more likely to cover the space of possibilities. In the remainder of this discussion we describe the influence of dynamics considerations on a motion representation and sketch possible implementations of its semantics.

SUMMARY

The need for better animation control is apparent from the literature. The qualitative factors of Effort-Shape notation are being used to suggest extensions to existing movement representations in directions consistent with known characteristics of human movement and conventional animation. We show how the motion qualities may be at least approximated by a combination of kinematics and dynamics computations, with kinetic control modulated by acceleration and decelerations derived from existing interpolation methods. In addition, the interactions between two motions by phrasing are handled explicitly by modifications expressed in the representation. Temporal, spatial, and relationship interactions may be described and executed within an appropriately detailed model.

Several animation systems are running or are under development at the University of Pennsylvania to demonstrate the feasibility and efficacy of these approaches. We are anxious to experiment with them and produce animations showing processes involving the interaction of several people with a complexity not yet demonstrated elsewhere.

ACKNOWLEDGEMENTS

This investigation has been greatly aided by discussions with many colleagues and students, especially Bonnie Webber, Jim Korein, Jon Korein, Steve Platt, Paul Fishwick, and Scott Steketee. Moral support from Ann Hutchinson Guest on issues of dynamics and Effort-Shape Notation is much appreciated. Special thanks to Suzanne Morin for the résumé.

This research is partially supported by NASA Contracts NAS9-16634 and NAS9-17239, NSF CER Grant MCS-82-19196, and ARO Grant DAAG29-84-K-0061.

The full text of this paper is available as a Technical Report from the author's department.

REFERENCES

1. Akita, Koichiro. Analysis of body motion image sequences. Proceedings of the 6th International Conference on Pattern Recognition, October, 1982, pp. 320-327.
2. Allen, James F. "Towards a General Theory of Action and Time". *Artificial Intelligence* 23, 2 (1984).
3. Armstrong, W. W. and Mark Green. The dynamics of articulated rigid bodies for purposes of animation. Proc. Graphics Interface '85, Montreal, 1985, pp. 407-416.
4. Badler, Norman I. *Temporal scene analysis: Conceptual descriptions of object movements*. Ph.D. Th., Dept. of Computer Science, University of Toronto, Toronto, Canada, 1975.
5. Badler, Norman I., Stephen W. Smoliar, Joseph O'Rourke, and Lynne Webber. The simulation of human movement. Dept. of Computer and Information Science, Univ. of Pennsylvania, Philadelphia, PA, 1978.
6. Badler, Norman I., Joseph O'Rourke, Stephen Platt, and Mary Ann Morris. Human movement understanding: A variety of perspectives. Proc. AAAI Conf., Stanford, CA, 1980, pp. 53-55.
7. Badler, Norman I., Bonnie L. Webber, James U. Korein, and Jonathan D. Korein. TEMPUS. Proc. IEEE Trends and Applications Conference, 1983, pp. 263-269.
8. Badler, Norman I. and Ruzena Bajcsy. "Three-dimensional representations for computer graphics and computer vision". *Computer Graphics* 12, 3 (Aug. 1978), 153-160.
9. Badler, Norman I. and Stephen W. Smoliar. "Digital representations of human movement". *ACM Computing Surveys* 11, 1 (March 1979), 19-38.
10. Badler, Norman I., Joseph O'Rourke, and Bruce Kaufman. "Special problems in human movement simulation". *Computer Graphics* 14, 3 (1980), 189-197.
11. Baecker, Ronald. Picture-driven animation. Proc. AFIPS Spring Joint Comp. Conf., AFIPS Press, Montvale, NJ, 1969, pp. 273-288.
12. Bartenieff, Irmgard, with Dori Lewis. *Body Movement: Coping with the Environment*. Gordon and Breach, New York, 1980.
13. Brady, Michael, John M. Hollerbach, Timothy L. Johnson, Tomás Lozano-Pérez, and Matthew T. Mason (eds.). *Robot Motion: Planning and Control*. MIT Press, Cambridge, MA, 1983.
14. Burtynk, N. and M. Wein. "Interactive skeleton techniques for enhancing motion dynamics in key frame animation". *Comm. of the ACM* 19, 10 (Oct. 1976), 564-569.
15. Calvert, T., J. Chapman, and A. Patla. "The integration of subjective and objective data in the animation of human movement". *Computer Graphics* 14, 3 (July 1980), 198-203.
16. Calvert, T., Chapman, J., and Patla, A. "Aspects of the kinematic simulation of human movement". *IEEE Computer Graphics and Applications* 2, 9 (Nov. 1982), 41-50.
17. Catmull, Edwin. A system for computer generated movies. Proc. ACM Annual Conf., 1972, pp. 422-431.
18. Clark, James. "Hierarchical geometric models for visible surface algorithms". *Comm. of the ACM* 19, 10 (Oct. 1976), 547-554.
19. Dell, Cecily. *A Primer for Movement Description*. Dance Notation Bureau, Inc., New York, 1970.
20. Derby, Stephen. "Simulating motion elements of general-purpose robot arms". *International J. of Robotics Res.* 2, 1 (Spring 1983), 3-12.
21. Emmett, Arielle. "Digital portfolio: Tony de Peltrie". *Computer Graphics World* 8, 10 (October 1985), 72-77.
22. Fishwick, Paul A. *Hierarchical Reasoning: Simulating Complex Processes over Multiple Levels of Abstraction*. Ph.D. Th., Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1986.
23. Gangel, Jeffrey S. A motion verb interface to a task animation system. Master Th., Dept. of Computer and Information Science, Univ. of Pennsylvania, August 1985.
24. Ginsberg, Carol M. and Delle Maxwell. Graphical Marionette. Proc. ACM SIGGRAPH/SIGART Workshop on Motion: Representation and Perception, April, 1983, pp. 172-179.
25. Girard, Michael and A. A. Maciejewski. "Computational modeling for the computer animation of legged figures". *Computer Graphics* 19, 3 (1985), 263-270.
26. Herbison-Evans, Don. "NUDES2: A Numeric Utility Displaying Ellipsoid Solids". *Computer Graphics* 12, 3 (Aug. 1978), 354-356.

27. Hollerbach, John M. "A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity". *IEEE Trans. Systems, Man, and Cybernetics* 10, 11 (Nov. 1980), 730-736.
28. Hollerbach, John M. Dynamic scaling of manipulator trajectories. A.I. Memo No. 700, Artificial Intelligence Lab, MIT, Cambridge, MA, January, 1983.
29. Hutchinson, Ann. *Labanotation*. Theatre Arts Books, New York, 1970.
30. JML Research, Inc. IMP-84. Madison, WI.
31. Klima, Edward S. and Ursula Bellugi. *The Signs of Language*. Harvard Univ. Press, Cambridge, MA, 1979.
32. Kochanek, Doris H. U. and Richard H. Bartels. "Interpolating splines with local tension, continuity, and bias control". *Computer Graphics* 18, 3 (1984), 33-41.
33. Korein, Jonathan D., and Norman I. Badler. "Temporal anti-aliasing in computer generated animation". *Computer Graphics* 17, 3 (July 1983), 377-388.
34. Korein, James U.. *A Geometric Investigation of Reach*. MIT Press, Cambridge, MA, 1985.
35. Korein, James U. and Norman I. Badler. "Techniques for goal directed motion". *IEEE Computer Graphics and Applications* 2, 9 (Nov. 1982), 71-81.
36. Lee, Hsi-Jian and Zen Chen. "Determination of 3D human body postures from a single view". *Computer Vision, Graphics and Image Processing* 30, 2 (1985), 148-168.
37. Lee, C. S. George. "Robot arm kinematics, dynamics and control". *IEEE Computer* 15, 2 (Dec. 1982), 62-80.
38. Linehan, Thomas E. "Ohio State pioneers computer animation". *Computer Graphics World* 8, 10 (October 1985), 48-60.
39. Loomis, Jeffrey, Howard Poizner, Ursula Bellugi, Alynna Blakemore, and John Hollerbach. "Computer graphic modeling of American Sign Language". *Computer Graphics* 17, 3 (July 1983), 105-114.
40. Lozano-Pérez, Tomás and Michael Wesley. "An algorithm for planning collision-free paths among polyhedral obstacles". *Comm. of the ACM* 22, 10 (Oct. 1979), 560-570.
41. Magnenat-Thalmann, Nadia and Daniel Thalmann. *Computer Animation: Theory and Practice*. Springer-Verlag, New York, 1985.
42. Marr, David and H. K. Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. Proc. Royal Soc. London B200, 1981.
43. Mezei, L. and A. Zivian. ARTA, an interactive animation system. Proc. IFIP Congress, 1971, pp. 429-434.
44. Michotte, A.. *La Perception de la Causalité*. Louvain, 1946.
45. Miller, George A. English Verbs Of Motion: A Case Study In Semantics And Lexical Memory. In *Coding Processes In Human Memory*, Arthur W. Melton and Edwin Martin, Ed., V.H. Winston & Sons, Washington, D.C., 1972, ch. 14, , pp. 335-372.
46. O'Rourke, Joseph and Norman I. Badler. "Model-based image analysis of human motion using constraint propagation". *IEEE Trans. PAMI* 2, 6 (Nov. 1980), 522-536.
47. Orlandea, N. and D. A. Calahan. A sparsity-oriented approach to the design of mechanical systems. In *Problem Analysis in Science and Engineering*, Academic Press, New York, 1977.
48. Parke, Frederic. "Parameterized models for facial animation". *IEEE Computer Graphics and Applications* 2, 9 (Nov. 1982), 61-68.
49. Paul, Richard. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, MA, 1981.
50. Paul, Richard P. "Manipulator cartesian path control". *IEEE Trans. Systems, Man, and Cybernetics* 9, 11 (Nov. 1979), 702-711.
51. Paul, Burton and Ronald Schaffa. DYSFAM User's Manual. Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania.
52. Platt, Stephen and Norman I. Badler. "Animating facial expressions". *Computer Graphics* 15, 3 (Aug. 1981), 245-252.
53. Platt, Stephen. *A Structural Model of the Human Face*. Ph.D. Th., Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1985.
54. Reeves, William T. "Inbetweening for computer animation utilizing moving point constraints". *Computer Graphics* 15, 3 (Aug. 1981), 263-269.
55. Robbins, D. H., R. O. Bennett Jr., and B. Bowman. User-oriented mathematical crash victim simulator. Proc. 16th Stapp Car Crash Conf., 1972, pp. 128-148.
56. Schank, Roger C.. *Conceptual Information Processing*. North Holland, Amsterdam, 1975.
57. Shelley, Kim L. and Donald P. Greenberg. "Path specification and path coherence". *Computer Graphics* 16, 3 (July 1982), 157-166.
58. Spegel, Marjan. Programming of mechanism motion. Tech. Report CRL-43, Div. of Applied Science, New York University, New York, 1975.
59. Sturman, David. Interactive key frame animation of 3-D articulated models. Proc. Graphics Interface '84, Ottawa, Canada, 1984, pp. 35-40.
60. Tsotsos, John, John Mylopoulos, H. Dominic Covey, and Stephen W. Zucker. "A framework for visual motion understanding". *IEEE Trans. PAMI* 2, 6 (Nov. 1980), 563-573.
61. Vere, Steven A. "Planning in time: Windows and durations for activities and goals". *IEEE Trans. on Pattern Analysis and Machine Intelligence* 5, 3 (May 1983), 246-267.
62. Waltz, David L. Event Shape Diagrams. Proceedings AAAI-82, 1982, pp. 84-87.
63. Weber, Lynne, Stephen W. Smoliar, and Norman I. Badler. An architecture for the simulation of human movement. Proc. ACM Annual Conf., Washington, D.C., 1978, pp. 737-745.
64. Wilhelms, Jane and Brian A. Barsky. Using dynamics for the animation of articulated bodies such as humans and robots. Proc. Graphics Interface '85, Montreal, 1985, pp. 97-104.
65. Zeltzer, David. Knowledge-based animation. Proc. ACM SIGGRAPH/SIGART Workshop on Motion: Representation and Perception, April, 1983, pp. 187-192.
66. Zeltzer, David. "Motor control techniques for figure animation". *IEEE Computer Graphics and Applications* 2, 9 (Nov. 1982), 53-59.

The Interactive Specification of Human Animation

G. Ridsdale, S. Hewitt and T. W. Calvert

Laboratory for Computer and Communications Research

Simon Fraser University, Burnaby, B.C. V5A 1S6

Abstract

This paper describes the Figure Animation Project in progress at Simon Fraser University. The project has two goals for the specification of figure animation: the first is to implement an interactive Figure Animation Test Bed for specifying movement at the *detail* level and the second is to develop a mechanism for describing figure animation at the *scene* level. These goals — and approaches to solving them — are discussed. In particular, the application of knowledge-based inference to the problem of scene description is examined.

Introduction

The long-term goal of this project is to develop scene-level motion descriptions for articulated, humanoid figures. In general, three-dimensional animation of the human body — or of any other vertebrate — is based on an underlying framework of articulated elements. For instance, a reasonable approximation of the human skeleton can be achieved with about 24 segments if the fingers and toes are ignored. Most of our efforts over the past few years have been directed towards developing interactive techniques for specifying detailed figure movements, since a system embodying such techniques is a necessary component for developing and testing scene descriptions. Such an interactive test bed must be capable of displaying the full range of interesting scene actions; this range includes both the actions of individual joints, and the movements of the body as a whole.

Above the detail level are the motivations of the characters and their interactions with the environment. This is animation at the *scene* level. After about ten years of continuous research in this area we are still not sure whether truly convincing animation of the full range of human movement is feasible, but there is no question that progress has been made in certain specialized areas of figure animation. What is needed now is the guidance of an intelligent supervisor to tie together these many pieces of the animation problem.

To coordinate the activities of a human animation system we require a supervisory program that has knowledge of the overall goals of the characters in the scene. In particular, such a program needs to deal with the issues of path planning and with the physical constraints on jointed figures. Ideally, it should only be necessary to specify the character's motivation in the scene and its initial position. The figure would then progress automatically and *characteristically* to its most likely destination from that starting point. In reality, the problem of determining an appropriate path from knowledge of the character's intentions is difficult, as is the problem of having the character navigate around both the fixed objects and the other characters in the scene. This latter problem has occupied robotics researchers for many years. The problem of sophisticated route planning is particularly difficult when the characters are jointed walking figures. Not only does the figure have to move about unencumbered, but the feet must also pick their way around and over any objects that may be found on the floor. While performing this, the figure must maintain its balance and a reasonable posture. This is a particular problem when shifting weight smoothly from one limb to another.

There are many physical constraints on what a character can and cannot do in a scene. For instance, a real figure's anatomy and physiology impose limitations on how it can move and interact. An intelligent supervisory program for human figure animation must understand these limitations; in this way, only physically realizable scenes will result.

Who Uses the System

The Figure Animation Test Bed is used mostly by choreographers who work in the disciplines of skating and dance. This test bed system has been designed to evaluate the various interactive techniques — buttons, menus, pick-and-drag, rendering speed traded off against image quality — that can be applied in a figure animation system. Since the people best qualified to judge the effectiveness of a system's user interface are themselves the future users, we select the techniques with which these choreographers feel most comfortable.

The major users of the scene-level animation system are expected to be film and theater directors. The intelligent supervisory program requires *expert knowledge* on which to base its decisions. This knowledge is developed through cooperation between the computer scientists who developed the system and the directors who possess the expert knowledge.

Project Background

Over the years, many techniques have been applied to the problem of specifying the complexity of human movement.

Rotoscoping

One basic approach is to capture real movement patterns from a live subject. This can be done by "rotoscoping" — digitizing by hand the joint co-ordinates of all body segments from at least two orthogonal views recorded on film or video. This approach, while accurate, is tedious. It is important in biomechanics research, and there is continuing interest in automating it, but the pattern recognition problems involved are difficult.

Live movement can also be captured in real time with special instrumentation. Goniometers provide a cumbersome but relatively inexpensive method [Calvert 80]. Expensive video scanning systems such as WATSMART and SELSPOT [Ginsberg 82], on the other hand, allow subjects to move freely in space; their actions are tracked by time-multiplexed light-emitting diodes attached to their joints. The movement patterns digitized with any of these methods can be normalized for speed and body size, and can be stored to create a library of fundamental movement patterns.

Notation

Another way movement patterns can be specified is with notation. While human movement can be described by a number of *dance* notation systems, such as as Labanotation [Smoliar 77] [Calvert 78] [Ryman 83], Eshkol-Wachman notation [Eshkol 79] and Benesh notation [Singh 83], none of these deal directly with the problem of describing human movement in an unambiguous way. This is because all of these systems are intended for use by trained dancers and, as a result, they normally leave out numerous details that these dance experts consider obvious. The knowledge these artists bring to bear on the interpretation of a Laban or Benesh score is an example of the sort of *expert knowledge* needed by any functioning movement interpreter, be it human or machine.

Our own experience with Labanotation has shown that it is a viable way to specify animation. Labanotation has

the definite advantage that it relies on the animators' conceptualization of the movements required, and it certainly lends itself to the development of complicated scores. However, the basic commands are at too low a level, and users have trouble predicting the outcome of commands. Even with the addition of a macro capability, it is still tedious (some might compare it to programming in assembly language). Not even dancers find it easy to learn.

These systems are capable of describing a movement in arbitrary detail. But since they lack the grammatical structure needed for the construction of higher-level primitives from simpler components, this capability is not enough. Thus the important characteristic of *extensibility* is missing (in any really useful form) from existing movement-notation systems.

Interactive Positioning

A third approach (after rotoscoping and notation), is interactive positioning. This is the basis of the Figure Animation Test Bed, and involves the interactive specification of body positions in a 3-D graphics environment. The user is presented with a space-filling vector representation of the human body on the screen of a graphics workstation. The body can be viewed from any angle — in perspective — under the control of a mouse or equivalent device. The mouse selects body segments and orients each segment in three-dimensional space. The end result is directly equivalent to the output from notation, but the user has direct visual feedback and finds the adjustments to be natural and intuitive.

Several attempts have been made to produce integrated systems for animating human movement [Calvert 80] [Calvert 82] [Calvert 83], [Badler 82] [Badler 79a], [Zeltzer 82] [Nichol 83]. Most of these have not directly addressed the problem of specifying the movement involved in a high-level, extensible way. Instead, the movement is described at what may be termed an "assembly-language" level, where it is difficult to collect (or "abstract") detailed movements together into complex actions. Although a simple form of *macro-expansion* is available in the system developed at SFU [Calvert 78], this still does not provide enough power to develop a complex hierarchy of movement concepts.

More fruitful work has been done in the area of the interface between an animation system and its users. Foley and Van Dam [Foley 82], describe the fundamental elements of good interactive design in terms of the *conceptual, semantic, syntactic* and *lexical* design levels.

The *conceptual* design level describes the *user model* of the system; that is, the key concepts that the user must

understand in order to make full use of the proposed system. The names and relationships of the *objects* in the system make up this level. In the context of our scene-level animation system, the conceptual level includes such things as characters, stage directions and props.

The *semantic* design level specifies the set of *functions* that the system is expected to perform. In our system, these functions include the ability to display a particular scene, the ability to learn new stage directions and character names, and so on.

At the *syntactic* design level, the rules that specify the acceptable sequences of input tokens (and the functions that they trigger), are set out. In the scene-level system, these values include the detailed protocol of menus, windows, valuator and text that will control the scene display and the management of the expert system.

Finally, the *lexical* level specifies the input tokens that the system recognizes. These include *text tokens* (such as "walk", "run" and "stage left"), *graphical tokens* (sketches and selected menu-items) and *gestural tokens* (movements of the user's body).

There is considerable interest in automating the development of a user interface, given its specification in some formal language. Such an automatic system is described by Olsen [Olsen 84]. Here a formal specification of a user interface can be used to generate a Pascal program that implements the functions of that interface. The input, in this case, is in the form of a *grammatical* description of the interface specification, which is entered by the system designer.

System Configuration

There are two prime requirements for an interactive environment in which the body positions are to be specified. The first is for realistic, three dimensional visualization of the spatial orientation of the figure; the second is for fast motion checking. To meet these requirements, we employ an IRIS 2400 Workstation as the heart of our hardware system configuration. A machine of this power, while expensive, is essential for smooth interactive positioning; a very large number of transformations is needed to represent two fully articulated figures. Hardware graphics power is also very important for fast and smooth motion checking. For these two needs, vector-based machines — such as those produced by IMI or Evans and Sutherland — could also have been used. However, in addition to fast line drawing and matrix computation capabilities, the IRIS contains a 32-bit frame buffer with both smooth shading and Z-buffer hidden surface elimination available in hardware. These

capabilities are useful while rendering the final animation.

Using the Figure Animation Test Bed

When the user develops a piece of choreography on the computer she performs four main steps. These steps are:

- to design the sequence of phrases.
- to interactively generate the keyframes.
- to interpolate the intermediate frames and finally,
- to motion test the result.

When the choreographer is satisfied, the resulting animated sequence may be rendered onto film.

Sequences

In our terminology, a piece of choreography is referred to as a *sequence*. A sequence in turn is composed of any number of *phrases*. A particular sequence is defined by a list of phrase names that are saved in a text file along with other global information that pertains to the sequence as a whole. This includes the number of phrases in the sequence and the number of frames in each phrase. Note that the same phrase may be re-used in any number of different sequences.

A phrase in turn is composed of a group of *keyframes*. Three keyframes are currently used to define each phrase since an approximating third-order spline is used to generate the intermediate positions. Each keyframe position is generated interactively on the screen of the IRIS 2400. At the start of this interactive process the user is presented with an image of two figures in a standard initial position. Each figure is represented by a vector image, where each body segment is modeled by a four-sided prism. It is important to use at least a crude space filling model like this, in order to give the user feedback on limb rotation and on the contact between body segments. Hidden lines are not removed. However, adjacent body segments are drawn with different colours to aid discrimination. The body parts themselves are positioned individually by a pick and drag procedure.

Interaction

To begin the interactive process, the user initially selects the foot which will act as the support point for the entire figure. Then the body position is built up by orienting each limb segment in turn, moving away from the support point. Using the mouse, a body segment is picked and its

three angles of orientation are adjusted in turn; the user can change the angle of view at will. A digital readout of the angles is given at the bottom of the display. A mouse-based valuator generates these analog values for the figure's joint angle orientations, which in turn determine the orientation of the body parts presented in real time on the screen of the system console. This provides immediate feedback to the system user. After the approximate body position has been achieved, the user will iteratively refine the inter-segment angles until the desired result is obtained.

To specify a second frame, the user can either start with a standard position, as she did with the first frame, or the first frame can be copied and used as the starting point for the second. Similarly a third frame is specified. These three frames form a phrase which is given a name and stored. Multiple phrases are built up in turn, typically using the last frame of one phrase as the first of the next. Very little typing need be done by the choreographer while interacting with the system. Instead, software buttons guide the user through the sequence of steps that result in the final animation.

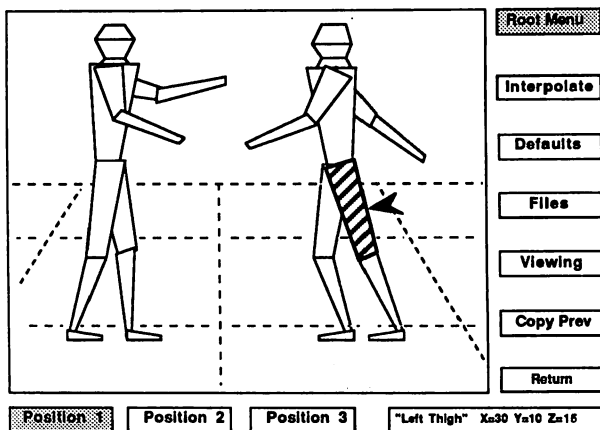


Figure 1 : Figure Animation Test Bed Screen

In this screen, the user has picked the left thigh of the right figure, and is about to change its orientation.

Currently, the user can specify the positions of two figures at a time. At this point the information available is equivalent to that which can be obtained by interpreting Labanotation commands or from instrumentation; thus data from different methods of movement specification can be combined.

Interpolation

Once the keyframes have been collected into phrases and the phrases assembled into a sequence, the "inbetweening" stage is performed. In this step a smooth series of frames connecting the keys is produced. A form of curve fitting based on third-order splines — one for each joint of the body — is used to determine the appropriate angle in space between the body parts articulated by that joint. Joint angle interpolation is very compute-intensive, requiring approximately 2000 floating point operations per frame. By formulating the cubic curve-fit in terms of matrix multiplication, the array-processing capabilities of the IRIS "Geometry Engines" can be used to solve the interpolation problem. Using these pipelined matrix processors, 1500 intermediate frames can be computed per minute.

Viewing

Having completed the interpolation, the final step is to view the resulting action. This involves selecting one of the display options that trade off rendering speed v.s. image quality. These rendering options include line drawings, filled polygons, and smoothly shaded solids.

Line Drawings This results in the fastest rendering. By drawing each body part as a rectangular prism, frame rates of about 4/sec can be achieved. Faster frame rates (as high as 15/sec) can also be achieved using a simpler stick figure.

Filled Polygons An intermediate level of rendering quality is obtained by using filled polygons to represent the body parts. This requires hidden surface elimination, increasing the rendering time to two seconds per frame.

Smooth Shading The highest quality rendering requires the use of smooth shading and an accurate lighting model. However, flexible, jointed figures produce special problems for any solid modeling technique; joint coverage is particularly difficult.

We have experimented with using spheres as building blocks for constructing the figures [Badler 79b]. The figures are built up using a Constructive Solid Geometry (CSG) approach where the primitives are shaded coloured spheres of varying diameter. There are over 800 spheres in each body and each sphere is rendered with a polygon approximation which takes account of the lighting for each sphere. The shading is obtained with the Gouraud method and the hardware z-buffer in the IRIS provides hidden

surface removal.

This resulting image contains over 400,000 smooth-shaded polygons per frame and requires 2-3 minutes to render on the IRIS 2400. Obviously, this is much too slow for previewing the motion of the figures, and so this technique is used only for frame by frame reproduction onto 16mm film.

Work In Progress

The facilities of the Figure Animation Test Bed system are being extended and the quality of figure rendering is being improved. Also, the design and programming of the scene-level animation system is under way.

Animation Test Bed

The test bed system has been used by a figure-skater and by a dancer as an experimental tool. This has already resulted in some significant segments of animation. Both users both have expressed a very definite preference for interactive specification over the use of Laban-style notation. As a result of their experience, two significant needs have been identified:

- The movement patterns in our animated films result from the smooth interpretation of keyframe body positions. One problem is that the result is *too* smooth to be credible as human movement. As a result, we are investigating methods of adding small oscillations to the interpolated data into the Test Bed system. Each new movement generates an oscillatory transient and a small "wobble" is present at all times. In this way the movements achieve a more natural quality.
- At present, the path traced out by a figure results from accumulating the individual actions (stepping, jumping, gliding) of the character. This makes higher-level path planning difficult. A higher-level route planning facility is being developed as part of the scene-level animation system.

Director's Apprentice

There is a long term interest in developing systems that can interpret very high level descriptions of animated scenes. A good real-world model of such a description is the *film script*. The knowledge needed to make sense of such a description requires the use of a *knowledge base* composed of *rules*. These rules will come from the knowledge and experience of an expert director. We have

named this project *The Director's Apprentice*, as it will "learn its craft" by studying the rules and practises of a human director. This learning process will take place as it aids him in the task of developing an interactive story-board.

The knowledge base of directing principles — used to interpret the script — will contain a collection of "if-then-else" rules. These rules map scene attributes — such as character motivations, script text and classic directing rules — into scene action. Of course, this mapping is neither unique nor well-defined, and it will result in numerous conflicting interpretations for each combination of attributes. A rule-interpreting inference engine must then arbitrate these conflicting conclusions, and decide on some reasonable resulting action for the scene. There are many categories of rules and concepts that such a system will need to address. These concepts include the inter-character feelings that dominate motivation, and the appropriate shifting of audience focus from one character to the next as the plot progresses. Knowledge of directing terminology — stage right, up stage, down stage — and standard set compositions will be needed to support the script interpretation.

Since considerable knowledge of the scene constraints is needed, the process of *abstracting* the detailed description of movement is not a trivial matter. For example, the phrase "*John walks across the room and stops at the other side of the desk*" may be used to describe many different scenes; the exact scene would depend on the arrangement of the furniture, on the other people in the room, on John's starting position and on his particular gait. To achieve the level of descriptive power needed for an effective directing language, an *expert* or *rule-based* system is being developed to provide the knowledge of scenes, physiology, habits, and so on, that are needed to abstract the directing concepts.

Expert systems in AI have been developed largely to solve problems involving the deduction of answers from a set of facts stored in a *knowledge base*. Such a knowledge base contains the accumulated knowledge of one or more experts in that particular field. But how does this differ from a conventional data-base management system? Perhaps, the most important capabilities that distinguish an expert system are:

1. **Inference:** the ability to form a long, complex conclusion using the information present in the knowledge base [vanMelle 81]. This knowledge base contains both *relations* — as in a conventional relational data base — and the *rules of inference* that permit the system to infer new relations from those that were initially given.

2. **Self-knowledge:** the ability of the system to understand and explain interactively the structure of its own data base and its own inferential mechanism [Davis 82] [Davis 80].
3. **Flexibility:** the ability of the knowledge base to grow and adapt to new knowledge as a result of correcting comments made by an expert consultant [Winston 81].

Expert-system projects have been developed for many problem domains, but only a few of these were ever taken to the point of being really useful [Nau 82]. Of these, probably the most successful implementation was the MYCIN [Shortcliffe 76] [Davis 82] project, a question-answering system intended for the problem of medical diagnosis. MYCIN was based on the use of production-system methodology, where the knowledge base consists of *productions* (or *rules*) of the form

"if A_1 is true,
and if A_2 is true,

and if A_n is true,

then C_i is true."

The applicability of production systems to the accumulation of expert knowledge has been discussed by Langely [Langely 83]. This work points out that the inherently *modular* nature of information that has been described in terms of productions makes the task of incorporating new information into the structure much easier. The considerable success of the MYCIN project in answering complex diagnostic questions — measured against the performance of real "expert" physicians — led to the development of EMYCIN [Davis 82]. This consists of the pure "expert system" of MYCIN, but stripped of the medical-diagnosis data base. It led to further successful tests of the production-system mechanism in other subject domains (such as civil engineering and geology [vanMelle 81]).

The success of production-based expert systems largely derives from the ability of these systems to *encapsulate* specific facts in the knowledge base in the form of individual productions [Davis 82], [Langely 83]. The advantages of this are that *local*, specific changes can be made to the knowledge base, (adding, modifying or deleting individual rules) and the changes' effect on the other rules in the system can generally be predicted in a straightforward manner. In fact, the same mechanism can also be applied to the strategies used by the reasoning system

itself, as described in [Davis 80]. This opens the possibility of having the search-strategy mechanism *itself* defined in terms of productions — which can be added, modified, and deleted easily.

The production system in MYCIN was augmented by its ability to assign a *certainty factor* to the conclusion of a rule. These "confidence measures" range from 100% ("is certain to") down to -100% ("is certain not to"). This involves having the rule interpreter combine the certainty factors of the subordinate rules to form the certainty of the conclusion. The use of continuous-valued (or *fuzzy*) logic in the inferential mechanism derives from the observation that few conclusions in any real domain can be made with absolute certainty [Zadeh 83]. Statements such as "If the character is John then there is an 80% chance that he will walk quickly across the room, and there is a 20% chance that he will walk slowly across the room," may be important when acquiring empirical judgements from human directors.

The function of an expert system is to answer questions using expert knowledge and reasoning. The "questions" that the Director's Apprentice will try to answer will arise during the interpretation of the script. For example the sentence "John walks quickly across the room, and stands behind the desk" will generate the following questions for the expert system to answer:

1. Is John presently in the room?
2. Are there any obstacles between John's present position and the desired final position?
3. If so, what is the best path for him to follow around, over, or under those obstacles? What sort of motion, if any, is meant by "walks quickly", or by "stands still"?

Issues of route planning, relating to the solution of problems (2) and (3), have been discussed by Lozano-Perez [Lozano-Perez 80]. More difficult questions relating to the *meaning* of the film, such as "Is it *in character* for John to kick over a chair on his way to the desk?" require the presence in the database of detailed information about the psychological and emotional aspects of the action. These considerations have been discussed by Fleischer [Fleischer 84].

Many recent expert systems have been concerned with the problem of using expert knowledge to make sense of sentences expressed in a natural language. Since we believe that free-form natural language is an inappropriate interface for an interactive graphics system, the linguistic issues that these systems address (pronoun references, multiple-clause sentences, and others) are not directly

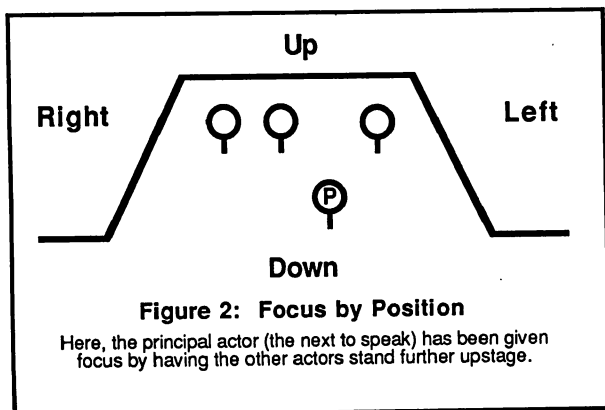
relevant to this research. Instead of an unconstrained natural language interface, we are implementing a *system-directed* conversation, implemented by menu-picking. The advantages of this approach have been described by Rich [Rich 84].

Interacting with the Director's Apprentice

Interaction with the Director's Apprentice involves two steps. The first step is to design a set of rules that capture the facts and relations inherent in the key directing concepts — this forms the knowledge base. The second step is to query the knowledge base using the control panel of the Director's Apprentice inference engine.

Most stage action in a play consists of explicit commands to the actors, such as "exit stage left", "approach the upstage character" and so on. This is termed *inherent* movement. However, stage actions may often be *inferred* from a script even if that script contains only dialogue. This is termed *imposed* movement. Many excellent texts (for example Allensworth [Allensworth 82]) have been written for theater students, detailing how certain types of action may be generated from a knowledge of the flow of dialogue from one speaker to the next.

The initial set of rules that have been tried in the knowledge base of the Director's Apprentice deal with the theatrical concept of *focus*. Focus concerns the use of subtle character action to shift audience attention from one character to the next, generally one step ahead of the next change of speaker. This is done so that the audience will have time to settle its "focus" on a character *before* he begin to speak; otherwise, his first few words or gestures may be lost to those of the audience who are watching someone else. The following are some ways that may be employed to achieve shift of focus, based on the known flow of dialogue in a script.



In *focus by position*, the principal actor — that is, the

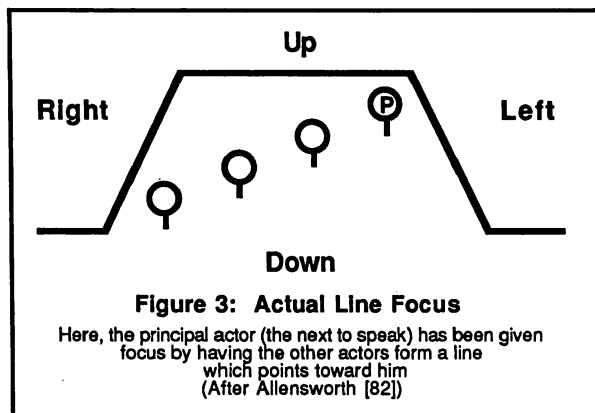
actor who is just about to begin speaking — receives focus by walking downstage, or by having the other actors walk upstage. This puts him in an attention-grabbing downstage position relative to the others. Encoded as a knowledge-base, this action may include such rules as:

IF next-to-speak is actor_i,

AND actor_j is-downstage-of actor_i,

THEN *j* moves-upstage-of *i*.

Many other rules, defining the concepts "is-downstage-of" and "moves-upstage-of" would also be required.



In *actual line focus*, the principal actor receives focus by having the other actors align themselves in a virtual "arrow" that points at him. To handle actual line focus, the knowledge base would need to contain rules such as:

IF next-to-speak is actor_i,

AND actor_j is-not-aligned-to actor_i,

AND actor_j is-closest-to actor_i,

THEN align-actor *j* *i*,

AND next-to-be-aligned-is *j*

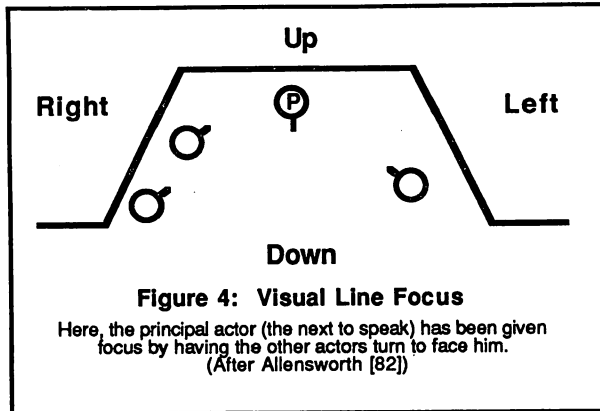
IF next-to-be-aligned is actor_k,

AND actor_l is-closest-to actor_k,

THEN align-actor *l* *k*,

AND next-to-be-aligned-is *l*

and so on.



In *visual line focus*, the principal actor receives focus by having the other actors turn to face him. To handle this focusing rule, the knowledge base would need to contain rules such as:

IF next-to-speak is actor_i,

AND actor_j is-not-facing actor_i,

AND actor_j is-closest-to actor_i,

THEN turn-to-face-actor *j,i*,

AND next-to-face-is *j*

IF next-to-face is actor_k,

AND next-to-speak is actor_i,

AND actor_i is-not-facing actor_k,

AND actor_i is-closest-to actor_k,

THEN turn-to-face-actor *i,k*,

AND next-to-face-is *i*

At present, an ordinary text editor is used to enter these rules into the knowledge base. A structured rule editor is planned.

The inference engine for the Director's Apprentice runs on a SUN Workstation, with a high-resolution (1150 X 890) bit-mapped screen. The various functions of the rule interpreter and query system are invoked by menu picks on a series of adjustable panels (windows) that pop up under control of the inference mechanism. This way, it is hoped that directors using the system will adapt quickly to the interactive dialogue.

Improved Rendering

The current rendering technique, based on a body built up from spheres, leaves much to be desired. The use of a skin approximated with a polygon mesh promises both the improved application of modern lighting models and smoother surfaces. However, when human figures are rendered by existing polygon modelers, the results are generally stilted and cartoon like. Closeups of bending joints are particularly difficult to render smoothly. A robust method to move the control points of a skin derived by splines is being developed. The goal is to achieve a skin which stretches naturally as the body segments move relative to each other.

Conclusions

Many areas of research need to be explored further, at both the detail level and scene level. At the detail level, more study needs to be done on the best ways to interactively specify figure movement. The present system is continually being revised in response to user comments and suggestions.

At the scene level, a great many more rules will be needed to effectively implement change of focus. Other forms of imposed action are being studied as well. In order to put the knowledge base on a firm theoretical foundation, we are also looking into developing a formal semantic model (as described in DelGrande [Delgrande 86]) of directing concepts.

Acknowledgements

We wish to thank Catherine Lee, our principal choreographer, for her comments in developing the Animation Test Bed. This research was supported, in part, by grants from NSERC and from the B.C. Science Council.

References

- [Allensworth 82] Carl Allensworth.
The Complete Play Production Handbook(Rev. Ed.).
Harper and Row, Inc., 1982.
- [Badler 79a] N.I. Badler and S.W. Smoliar.
Digital representation of human movement.
Computing Surveys 11:19- 38, March, 1979.
- [Badler 79b] N.I. Badler, J. O'Rourke and H. Toltzis.
A spherical representation of a human
body for visualizing movement.
Proc. IEEE 67:1397- 1402, 1979.
- [Badler 82] N.I. Badler.
Modelling the Human Body for
Animation.
IEEE Computer Graphics and Applications
2, November, 1982.
Special Issue.
- [Calvert 78] T.W. Calvert and J. Chapman.
Notation of movement with computer
assistance.
In *Proceedings Annual Conference*, pages
731- 736. ACM, 1978.
- [Calvert 80] T.W. Calvert, J. Chapman and A. Patla.
The integration of subjective and objective
data in animation of human
movement.
Computer Graphics 14:198- 203, 1980.
- [Calvert 82] T.W. Calvert, J. Chapman and A. Patla.
Aspects of the Kinematic Simulation of
Human Movement.
IEEE Computer Graphics and Applications
2:41- 50, November, 1982.
- [Calvert 83] T.W. Calvert.
Computer assisted filmmaking: A review.
In *Proceedings of Graphics Interface 83
Conference*. Graphics Interface,
Edmonton; May 1983.
- [Davis 80] Randall Davis.
Meta-Rules: Reasoning About Control.
Technical Report AI Memo No.576, MIT
AI Lab, 1980.
- [Davis 82] Randall Davis and Douglas Lenat.
*Knowledge-Based Systems in Artificial
Intelligence*.
McGraw-Hill, 1982.
- [Delgrande 86] James P. Delgrande and John Mylopoulos.
Knowledge Representation: Features of
Knowledge.
*Fundamentals in Man-Machine
Communication: Speech, Vision and
Natural Language*.
Cambridge University Press, 1986.
- [Eshkol 79] Noa Eshkol.
Movement Notations.
Movement Notation Society, Tel Aviv
University, Tel Aviv, Isreal, 1979.
- [Fleischer 84] Kurt Fleischer, Mark Vickers, Ann
R. Marion, James R. Davis.
Towards Expressive Animation for
Interactive Characters.
In *Proceedings of Graphics Interface 84*.
Graphics Interface, Ottawa, Ontario,
May 1984.
- [Foley 82] James D. Foley and Andries Van Dam.
*Fundamentals of Interactive Computer
Graphics*.
Addison-Wesley, 1982.
- [Ginsberg 82] Carol M. Ginsberg and Delle Maxwell.
*Graphical Marionette: A Modern-Day
Pinocchio*.
Technical Report, Architecture Machine
Group, MIT, 1982.
- [Langely 83] Pat Langley.
Representational Issues in Learning
Systems.
IEEE Computer , October, 1983.
- [Lozano-Perez 80] Tomas Lozano-Perez.
*Automatic Planning of Manipulator Transfer
Movements*.
Technical Report AI Memo No.606, MIT
AI Lab, 1980.
- [Nau 82] Dana S. Nau.
Expert Computer Systems: A Tutorial.
Technical Report TR-1201, Computer
Science Dept., University of Maryland,
College Park, Maryland 20742, August,
1982.
- [Nichol 83] C.J. Nichol.
Animation Systems for Rigid Bodies.
Technical Report, Computing Science
Department, Glasgow University,
Glasgow Scotland, 1983.

- [Olsen 84] Dan R. Olsen and Elizabeth P. Dempsey.
SYNGRAPH: A Graphical User Interface
Generator.
In *SIGGRAPH-83*. ACM, Detroit,
Michigan, July, 1984.
- [Rich 84] Elaine Rich.
Natural-Language Interfaces.
IEEE Computer, September, 1984.
- [Ryman 83] R. Ryman, A. Patla and T. Calvert.
Use of Labanotation for clinical analysis
of movement.
In *Conference*. International Congress
Kinetography Laban, August, 1983.
- [Shortcliffe 76] Edward Shortcliffe.
*Computer-Based Medical Consultations:
MYCIN*.
American Elsevier, New York, 1976.
- [Singh 83] A. Singh.
A computerized editor for Benesh
movement notation.
Master's thesis, University of Waterloo,
1983.
- [Smoliar 77] S.W. Smoliar and L. Weber.
Using the computer for a semantic
representation of Labanotation.
Computing and the Humanities.
University of Waterloo Press, Waterloo,
Ontario, 1977, pages 253-261.
- [vanMelle 81] William J. van Melle.
*System aids in constructing consultation
programs*.
University Microfilms International, 1981.
- [Winston 81] Patrick H. Winston.
*Learning New Principles from Precedents
and Exercises: The Details*.
Technical Report AI Memo No.632, MIT
AI Lab, 1981.
- [Zadeh 83] Lotfi A. Zadeh.
Commonsense Knowledge Representation
Based on Fuzzy Logic.
IEEE Computer, October, 1983.
- [Zeltzer 82] D. Zeltzer.
Motor control techniques for figure
animation.
IEEE Computer Graphics and Applications
2:53-59, November, 1982.

Goal Directed Animation using English Motion Commands

Karin Drewery

John Tsotsos

Dept. of Computer Science

University of Toronto

ABSTRACT

This paper describes a prototype 3D animation system which can execute limited types of english motion commands by solving simple goals and directions.

This system has a frame-based knowledge base (KB) to describe objects and another to describe motions. A hierarchical planning system uses the motion descriptions as forward production rules to form a plan for a goal task. Directional commands relative to the object or a reference object can also be processed by referring to the directional description in the motion KB and the object's reference frame.

The underlying objective is to develop a method to incorporate goals into a graphical animation system so that it will be a *task level* system [Zelt85], where a behaviour is described in terms of events and relationships and frees the user from specifying all details of a motion.

Keywords : KB graphics, animation, motion description, goals.

1. Background

Developing animation systems which are task level systems is a relatively new research area in computer graphics and a complete one does not yet exist. Currently some *animator* level systems [Zelt85] in which the behaviours of the objects are described algorithmically in a programming language, exhibit properties that would be useful to a task level system.

In Reynold's actor-based [Reyn78] and Murtagh's object-based [Murt85] systems, objects can pass messages which allows adaptive motion. Adaptive motion occurs when the control processor uses information about the objects and their environment to control the objects' movements [Zelt85]. In MIRA [Thal83, Magn84, Magn85] attributes of the objects can be updated and examined also allowing adaptive motion.

Badler [Badl80, Badl81, Badl82] has been involved in representing human motion and developed Tempus. This system contains resolved motion algorithms for limb positioning and approaches task level animation. Zeltzer [Zelt82, Zelt83] developed SAS which uses local motor primitives (LMP's) to execute movement functions which have preconditions and has described ideas for a KB system which would be a task level animation system.

In the meantime work in Artificial Intelligence concerning motion descriptions was being done. Using Miller's [Mill72] classification of motion verbs, Badler

[Badl75] and Tsotsos [Tsot80] added their own modifications to create motion description and analysis systems. The KB for our system GEMS was derived from a modification of the frame based system proposed by Tsotsos.

A frame [Mins75, Gold79] is a representational structure which consists of slots which can describe parts of the item being represented and can contain information describing the relations between the slots. The slots form a PART-OF hierarchy. The frames form an IS-A hierarchy defining general to specific information (see example 1).

A frame based system was proposed because it can express hierarchical descriptions, allows general to specific levels of detail and conveys inheritance of properties. A frame which is below another frame in the IS-A hierarchy inherits the properties of the one(s) above it.

GEMS arose by incorporating a motion processing queue scheme similar to Zeltzer's [Zelt82] to process the information in the frames of the KB.

2. Overview

The user must first define a KB for 3D hierarchical objects and a KB for their motion descriptions. To begin an animation the user must *instantiate* objects from the frames defined in the object KB.

The next phase involves a motion processor which accepts motion commands in this format:

subject motion-verb reference | directional adverb
or
agent motion-verb subject reference | directional adverb

The command is parsed and existence checks on the objects are done by searching the object KB.

The task manager then consults the motion KB to process the motion verb. This is done by traversing the hierarchies described in the frame structured KB. From the hierarchical descriptions, the motion verb is broken down into its underlying primitives, which are internally defined procedures. If the motion has preconditions then the planning system may be called.

The planning system is modelled after ABSTRIPS [Saxe74,Nils80] a simple hierarchical planning system. A hierarchical planner was chosen because in many situations a subgoal condition of a goal can be regarded as a detail and does not need to be solved until the major steps of the plan are solved. Thus the plan is developed level by level.

A goal directed motion task in the KB will have a precondition list with priorities assigned to each precondition to indicate which ones should be solved first. A precondition is an action or state which must be executed before the task can be done.

The task will also have *delete* and *add* state lists. The *delete* list refers to states which are no longer true after the task has been completed and the *add* states are those which are now true. These lists are used by the planner to keep track of the current state situation.

The planner begins with the initial state situation and applies the appropriate motion tasks, which are actually forward production rules, to achieve the desired goal. A task is selected to be a possible element of the plan if a state in its *add* list matches the top of the goal stack. If this task has preconditions whose priorities are greater or equal to the current maximum priority value then they are placed on the stack. Otherwise the current state description is updated by removing states which are in the task's *delete* list and adding those listed in the *add* list. The chosen sequence of motion tasks forms a plan to achieve the original goal. Details of the planning algorithm can be found in the references mentioned.

The user may specify precisely which motion will supply the precondition needed, by using an *if* statement. Or instead, the user may specify only the states that are preconditions and the planner will determine the motions that will achieve these states. Using the plan and hierarchical information from the KB, a motion queue for each object that was referenced in the command is built.

A clock is run and at evenly spaced time intervals checks are done on each queue to determine which motion, if any the object should be undergoing. Interaction conditions are evaluated as the motion is executed

and may cause nodes to be added or removed from the queue. During this, current state information must also be updated. For each frame of the animation a file containing the appropriate transformations and object data is created and can later be passed to a rendering system.

3. Object Description

An object frame consists of a *description* and a *dependent* section. The *description* section contains slots which define an object's parts. A part has a name and a type which is either another user defined frame in the object KB or a system defined object primitive such as a cube, a sphere, a vector etc. If the object's parts are joined then the joint may be specified by the use of a *connected-to* expression. Constraints on the rotation angles can be defined. A slot may also contain constraints on dependent variables.

An object's geometry type is indicated by a slot type or in a "IS-A" expression. The type can be a 3D primitive or a user defined polyhedra or mesh, both of which can be defined in the KB or input from a file.

The *dependent* section contains definitions for variables which are dependent on the object's structure. An object in this system must have a centroid, a bounding box and direction vectors. Direction vectors are formed from the centroid to each face of the bounding box to discern the top, bottom, left, right, front and back of an object.

Example 1 shows some of the frames which could be used to describe a robot with joints. Notice that the constraints on the slots appear between the square brackets.

4. Motion Description

A motion frame consists of a *description*, *dependent*, *preconditions*, *interactions*, *delete* and *add* state sections. The *description* section (similar to an object's) contains slots which describe the parts of the motion. Each slot contains a name or label for the part followed by a type which is either another user defined motion from the KB or a primitive motion type (rotate, translate, scale). This may be followed by constraints on the type's dependents such as timing or speed.

There is also a *subj* slot to allow the user to define the type of object that exhibits the motion. An *agent* slot allows the user to specify the type of object that produces the motion. Similarly there is a *ref* slot where the user can define the type of object (if any) that the motion references.

The *dependent* section contains slots which define the parameters or variables of a motion. The label of the slot is the dependent's name followed by its type which must be a system defined primitive.

The *preconditions*, *add* and *delete* lists are used by the planning system as mentioned. A precondition allows the user to specify in the KB, which motions or

states must be done before the desired command can be executed. Suppose that the command "RobotA EXIT roomB" was given. Example 2 describes a frame for *EXIT* and specifies its preconditions.

In this example the preconditions are listed as states. The numbers in the labels indicate their priorities. First RobotA must be inside roomB as indicated by p1. The second precondition requires that the room have a door, otherwise we shall assume that the robot cannot exit the room. Then, in order to exit the room the robot must be standing and he must be near the door. A motion frame such as *APPROACH* would direct the robot to the door. The last precondition requires that the door be open.

The action of exiting the room is done by the frame *WALK_THROUGH* specified in the motion description. Notice that after the robot has exited the room the states *OUTSIDE_OF* (*subj.ref*) and *STANDING* (*subj*) must be added to the current state list.

The user must create motion frames in the KB with many preconditions to create more realistic goal-oriented descriptions.

Interaction conditions are tested while an action is occurring. In Example 3, the frame description for *FLIGHT* of a missile rocket, the interaction condition is to check if the rocket contacts any object in space. If it does then it will explode. In this example the user specifies precisely which action will occur using the *if* statement rather than just specifying states as in the previous example.

5. Conclusions and Extensions

GEMS presents a method to execute goal-directed graphics by using an object and a motion KB and a simple planning algorithm. The *preconditions*, *add* and *delete* lists in the motion KB enable the system to form a plan for the motion. Internal procedures calculate directions and relationships, and perform graphical motion primitives needed to display the plans. It would be a more powerful system if it could define these internal procedures.

The planning system used is limited. It does not account for the interaction of many agents which is needed in some animations. It is based on state changes and assumes that any state not mentioned in the delete and add lists are unchanged. A more recent work by Stuart [Stua85] has developed the idea of synchronizing multi-agent activities.

To incorporate a larger class of goal oriented commands GEMS should be extended to contain a reach algorithm for jointed limbs [ORou78, Kore82] and a complex path planning algorithm [Loza79].

Example 1 Description of an Object

Frame ROBOT is-a 3D_LINKED_OBJECT, MOBILE

Description:

head: HEAD [body connected to head at (0,5,0)];

body: 3D_RECT [xwidth = 4.0;

ywidth = 6.0;

zwidth = 2.0];

right_leg: RIGHTLEG

[body connected to right_leg at (1,6,0),

rotx < 90, rotx > -90,

roty < 90, roty > -90,

rotz < 90, rotz > -90];

etc.

end Frame

Frame HEAD is-a ELLIPSOID

Description:

c1: [xradius = 3, yradius = 4, zradius = 3];

end Frame

Example 2 for "RobotA EXIT roomb"

Frame EXIT is-a 3D_SEQUENTIAL_MOTION

subj: ROBOT;

ref: ROOM;

Preconditions,Delete:

p1: INSIDE (subj,ref);

p2: PART_OF (ref,door);

p3: STANDING (subj);

p4: NEAR (subj,ref);

p5: OPEN (ref.door);

Description:

**d1: WALK_THROUGH [ref = ref.door,
dur = 2];**

Add: OUTSIDE_OF (subj,ref), STANDING (subj);

end Frame

Example 3 for " Missile FLIGHT to Moon "

Frame FLIGHT is-a 3D_SEQUENTIAL_MOTION

Interactions:

p1: If (CONTACTS(subj,ANY)) then EXPLODES;

Description:

subj: ROCKET;

ref: OBJECT_IN_SPACE;

launch: LAUNCH [start_time = 5,

duration = 30,

speed = 100 units/sec];

phase1: PHASE1 [duration = 90,

speed = 200 units/sec];

phase2: PHASE2 [duration = 50,

speed = 300 units/sec];

end Frame

BIBLIOGRAPHY

- Badl75** Badler, N., "Temporal Scene Analysis: Conceptual Description of Object Movements", PHD Diss., University of Toronto, Feb, 1975
- Badl80** Badler, N., O'Rourke, J., Kaufman, B., "Special Problems in Human Movement Simulation", IEEE Computer Graphics, vol 14, #3, 1980
- Badl81** Badler, N., "Understanding Human Movement Synthesis and Analysis", Proceedings of the Conference on Information and Systems, 1981
- Badl82** Badler, N., "Design of Human Movement Representation Incorporating Dynamics and Task Simulation", Sig'82 Tutorial, July, 1982
- Badl84** Badler, N., Korein, J., "TEMPUS: A Sytem for the Design and Simulation of Human Figures in a Task-Oriented Environment", Unpublished, Dept. of Computer
- Gold79** Goldstein, "Using Frames in Scheduling", AI: An MIT Perspective, Brown MIT Press, 1979, pg 256
- Hewi73** Hewitt, C., Bishop, P., Steiger, R., "A Universal Actor Formalism for AI", IJCAI, 1973, pg 235
- Kore82** Korein, J., Badler, N., "Techniques for Generating the Goal-Directed Motion of Articulated Structures", IEEE Computer Graphics, Nov. 1982, pg.71
- Loza79** Lozano-Perez, T., Wesley, M., "An algorithm for Planning collision free paths among Polyhedral objects", Communications of the ACM, Vol. 22, #22, Oct. 1979, pg 560
- Magn83** Magnenat-Thalmann, N., Thalmann, D., "The Use of 3D High-level Graphical Types in the MIRA Animation System", IEEE Computer Graphics and Applications, Vol. 3, No. 9, 1983, pg 9
- Magn85** Magenat-Thalmann, N., Thalmann, D., Fortin, M., "Miranim: An extensible Director-oriented system for the animation of realistic images", IEEE Computer Graphics and Applications, Vol. 4 #3, March 1985.
- Mill72** Miller, G., "English Verbs of Motion: A case study in semantics and lexical memory", ed. by Martin and Meltin, V.H. Winston and Sons, Washington, 1972.
- Mins75** Minsky, M., "A Framework for Representing Knowledge", Psychology of Computer Vision, McGraw-Hill, New York, 1975
- ORou78** O'Rourke, Joseph, "Three Dimensional Motion of a 3-link System", Movement Report No. 11, MS-CIS-78-31, University of Pennsylvania, 1978.
- Reyn82** Reynolds, C., "Computer Animation with Scripts and Actors", Proceedings of Sigraph, 1982, pg 289
- Stua85** Stuart, C., "An Implementation of a Multi-Agent Plan Synchronizer", Proc. IJCAI-85, vol. 2, 1985
- Thal83** Thalmann, D., Magenat-Thlamann, N., "Actor and Camera Data Types in Computer Animation", Graphics Interface, 1983
- Tsot80** Tsotsos, J., "A Framework for Visual Motion Understanding", PHD Diss., University of Toronto, 1980
- Zelt82** Zeltzer, D., "Motor Control Techniques for Figure Animation", IEEE Computer Graphics, Nov., 1982, pg 53
- Zelt82** Zeltzer, D., "A Motion Planning Task Manager for a Skeleton Animation System" Sigraph 1982 Tutorial, July, 1982
- Zelt83** Zeltzer, D., "Knowledge-Based Animation", ACM Sigraph/Sigart Inter-disciplinary Workshop on Motion, Toronto, 1983, pg 187
- Zelt85** Zeltzer, D., "Towards an Integrated View of 3-D Character Animation", Proc. Graphics Interface '85, May, 1985, pg 105

SPEECH AND EXPRESSION : A COMPUTER SOLUTION TO FACE ANIMATION

Andrew Pearce, Brian Wyvill, Geoff Wyvill, David Hill

Department of Computer Science, University of Calgary.
2500 University Drive N.W.
Calgary, Alberta, Canada, T2N 1N4

Abstract

Animation which uses three dimensional computer graphics relies heavily on geometric transformations over time for the motions of camera and objects. To make a figure walk or make a liquid bubble requires sophisticated motion control not usually available in commercial animation systems.

This paper describes a way to animate a model of the human face. The animator can build a sequence of facial movements, including speech, by manipulating a small set of keywords. Synchronized speech is possible because the same encoding of the phonetic elements (segments) is used to drive both the animation and a speech generator. Any facial expression, or string of segments, may be given a name and used as a key element. The final animated sequence is then generated automatically by expansion (if necessary) followed by interpolation between the resulting key frames.

We present two alternative modelling techniques for constructing the face: a polygon mesh and a functional description using a technique called *soft objects*.

Résumé

L'animation par ordinateur en 3-dimensions compte fortement sur des transformations géométriques sur temps pour les mouvements de la caméra et des objets. Pour faire marcher une silhouette ou bouillir une liquide, il faut un contrôle de mouvement sophistiqué, qui n'est pas généralement disponible aux systèmes d'animation commerciaux.

Cet article fera la description d'une façon d'animer un modèle d'un visage humain. L'animateur peut construire une séquence de mouvements faciaux, y compris le discours, en manipulant une petite série de mots clés. Le discours est automatiquement synchronisé, parce que les mêmes éléments phonétiques contrôlent l'animation et la restitution vocale. N'importe quelle expression faciale, ou série de segments, peut être nommée, et utilisée comme élément clé. En dernier lieu, la séquence d'animation finale est créée automatiquement par l'expansion (si nécessaire) suivi par l'interpolation entre les images clés.

Nous présenterons deux techniques alternatives de construire le visage: à polygonale maille, et par une description fonctionnelle utilisant des *objets mous*.

Introduction

Three dimensional animation using computer graphics often suffers from a lack of sophisticated motion. Current modelling techniques can produce realistic looking images, but are not suited to representing objects in motion. Nor do we have established ways to describe complex motion to the computer system.

The human face is a prime example of an object which moves in a very complicated way, that cannot be easily and convincingly controlled by simple geometric transformations in time, unless constraints are placed on the possible motion. The major work in this area was done by Fred Parke at the University of Utah [Parke 74] and later developed at New York Institute of Technology [Parke 82]. Parke uses a face built from polygons and identifies groups of polygons which can be changed according to a set of parameters to control facial expressions and features. A second approach [Platt 81] is to use a structure based model where the muscles to be moved are described. While simulating the underlying facial muscles allows for exact representations of wrinkles and face motions, an adequate facial model has not been fully developed using this representation. This is due both to the difficulty in encoding all of the facial muscles and the complexity of its motion due to the number of degrees of freedom allowed the animator.

Although much work has been done on the modelling of the face, synchronized speech animation is still effected through rotoscoping or related techniques.

The Graphicsland Animation System

The *Graphicsland* project group [Wyvill, B. 85a] at the University of Calgary has developed an organised collection of software tools for producing animations from models in three dimensions. The system allows the combination of several different kinds of modelling primitive [Wyvill et al 85b]. Thus polygon based models can be mixed freely with fractals [Mandelbrot 83, Fournier 82] and particles [Reeve 83] in a scene. Motions and camera paths can be described, and animations generated. Note that we do not include the use of a two dimensional "paint" system. Our objective is always to construct views of a full three dimensional model.

Our objective in this work was to introduce better techniques for motion control than commonly available and integrate them into *Graphicsland*.

Interfacing to the Parke Model

The face-representation we used has been developed from Fred Parke's work at the University of Utah [PARKE 74]. Parke models the face as a collection of polygons which may be manipulated through a set of 50 numerical parameters. These control such things as length of nose, jaw rotation, shape of chin, and other similar facial features, and allow movement of these features by interpolating the parameters.

To describe motion directly using these parameters is clumsy and difficult. Motions were described as a pair of numeric tuples which identified; the initial frame, final frame, and interpolation type, followed by the parameter, the parameter's initial value, and the final value. In order to aid the animator, a keyword based interface was developed. The interface makes it possible to build up libraries of partial expressions (smile and blink would be two partial expressions) and place them anywhere within an animated sequence. It also has the ability to detect conflict should two simultaneous partial expressions attempt to manipulate a facial feature in opposing directions at any point in the animation.

Expression

We specify each partial expression by means of a set of keywords. We must describe:

- 1) the part of the face to be moved (eyes, mouth, cheeks...),
- 2) the type of movement (open, arch, raise...),
- 3) the initial and final frame number
- 4) the parameter value (normalized) at the final frame,
- 5) and optionally, the type of interpolation (default is linear)

For example, to open the mouth the dialogue might be:

```
open mouth frame 12 25 value 0.8
```

This would cause the mouth to open with 80% (0.8) of the maximum jaw rotation, beginning at frame 12 and ending at frame 25 of the animation. Alternatively, motions may be grouped together into a key element, for example, a blink expression might be specified:

```
animate blink
    close eyes frame 1 2 value 0
    open eyes frame 2 3 value 0.9
end
```

Once an expression has been specified, the animator may place that motion at several places in the animated sequence:

```
add blink frame 25
add blink frame 36
```

Figures 1a,1b and 1c show three frames from the blink sequence. Figure 2,3 and 4 show various expressions. Figure 4 also has had hair grown on the head using a number of particle generators distributed on the polygons which define the scalp.

Speech

In all work so far, the animation of a talking sequence for one of these facial models has been done using a technique similar to roto-scoping. A human actor is filmed, reciting the required script, and the facial model is constrained to follow the sequence of lip and jaw positions needed for each frame of the animation.

This process is tedious and expensive. Various alternative approaches exist, all based on some knowledge of the relationships between speech sounds and the configuration of the articulators. Articulation models giving information about jaw position, lip spreading or rounding, tongue position, and their dynamic relationships, for various speech sounds, can be built up from the literature on lip reading and acoustic phonetics [Walther 1982, Levitan 1977]. Subjective evaluation of the adequacy of such models, followed by correction and re-evaluation ensures that the models are good functional representations. An animator can use such model data to set up key frames corresponding to successive segmental articulations. The computer can interpolate the key frames according to more or less simple rules, and the resulting product can be dubbed in the usual way. Alternatively, the articulatory parameters may be controlled by recognition of the sounds produced by an actor, which ensures natural rhythm for the resulting speech. However, speech recognition is still less than perfect, and such systems tend to rely on sound classification that is both crude and error prone. Our approach is to synthesise both the speech and the sequence of facial expressions by rules, based on the articulatory model for the facial movements, and based on rules for acoustic synthesis for the speech. Thus phonetic script incorporating both segmental and suprasegmental information drives both aspects of the speech animation sequence. Synthetic speech is still somewhat unnatural, but such speech animation is both intelligible and well synchronised. Large quantities of speech animation can be generated at virtually no more cost than the graphical animation that forms part of it. Furthermore, script changes can be incorporated without having to rely on the availability of a particular real speaker. The synthesis is based on a long-standing research project that includes a new model for speech rhythm based on a generalisation of real speech rhythm data [Hill 1978a, Hill 1978b].

Normal speaking rates vary a good deal. Typical segment (speech sound) durations for normal speech vary between 50 and 250 milliseconds. Each articulation changes in a basically continuous fashion into the next one. For the acoustic synthesis, piecewise linear interpolation of the acoustic parameters, from one target to the next, according to relatively simple rules a few time divisions, has been found adequate for high quality synthesis. A typical sampling rate would be one sample every 10 milliseconds, but linear changes over periods of 20 to 80 milliseconds occur. A similar approach is being adopted for the interpolation needed for the changing facial expression dictated by the moving articulators. The rate of interpolation varies, just as for the acoustic synthesis (and in synchrony with it), but the rules are few and simple. At 24 frames per second, the average frame rate for a movie is approximately one frame every 40 milliseconds. This is well matched to the sampling rate needed for a fairly accurate representation of the synthetic speech, as might be expected from observations of real speech on film.

As an example of speech, the phrase "Hi there" could be achieved as follows:

```
sentence greet
    h
    ah
    i 170 0.3
    th
    e
    r
end
```

A sentence is specified and denoted by the name "greet" in order to allow the user to refer to the sentence again for modification, deletion or placement. The "end" command marks the end of the sentence, and allows the system to calculate the number of frames required for the speech. The duration and enunciation parameters available for each segment are being used with the diphthong "AH I"; it's duration is now 170 ms. and it's enunciation in terms of mouth position is 0.3 of it's full possible range. If these parameters

are not supplied, the default duration for each element of the phonetic script is used.

Once the animator is satisfied with the placement of the partial expressions, the sequence may be examined for motion conflicts with the command "check". Should a conflict be discovered, the partial expressions which clash are displayed and may be edited.

Figures 5a,b,c,d and e show selected frames produced from the sentence "greet".

Soft Objects

The term "soft object" is used to refer to the particular class of objects whose shape varies constantly because of forces imposed on it by its surroundings.

We have been experimenting with a general model for *soft* objects which represents an object or collection of objects by a scalar field. That is a mathematical function defined over a volume of space. The object is considered to occupy the space over which the function has a value greater than some threshold so the surface of the object is an iso-surface of the field function. That is a surface of constant function value within the space considered. The idea of using such surfaces for 3D modelling was first put forward by Jim Blinn [Blinn 82] and refined in the *Graphicsland* system [Wyvill 85c]. Using the field function developed in [Wyvill 85c], such surfaces can be finely controlled by varying the radius of influence and field value due to each key point. Our initial experiments suggest that fewer key points are needed to control the facial movements than with other techniques, and the process is computationally less expensive than using B-splines or polygon meshes.

Although a polygon mesh is a useful representation for the face model, it forms only a crude approximation to the smooth curves of a face, and suffers from the problem that current shading techniques smooth the centre the mesh leaving an un-smoothed polygon silhouette edge. B-spline patches [Huitric 85] have been used to define a smooth surface and fewer control points are needed to define the face than with polygons. However a set of soft object key points share these advantages and have several more. Soft object control points may have different colours associated with them. The colour of a control point affects the colour of a local region of the face, and this colour will be smoothly blended into the colours of the surrounding regions. A face may contain various areas of different colour, for example, rosy cheeks, red lips, a dark chin and pale forehead. The colours of control points can be made to vary with time, causing smooth changes of colour in selected parts of the face, as in a blush.

Conclusion

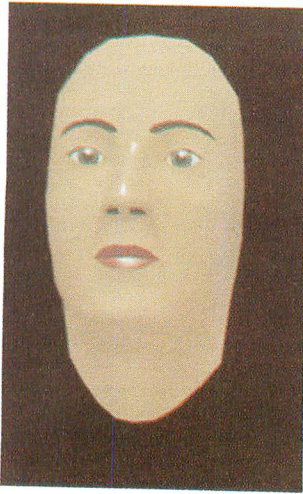
We have presented some experimental work with a face model. Different modelling techniques for facial animation have been described along with a method of using the model to produce synchronized animation directly from a speech synthesizer. The user interface to the speech and expression program is particularly simple and effective.

Acknowledgements

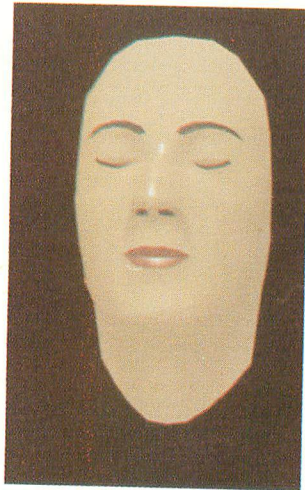
We would like to acknowledge the help and advice of our colleague Fred Parke, who supplied us with the original face data and has been extremely helpful to our work on *Graphicsland*. We also thank Milan Novacek for his help with the particle hair and Yung-Hsien Yen for his work on the user interface. The work is partially supported by the Natural Science and Engineering Research Council of Canada.

References

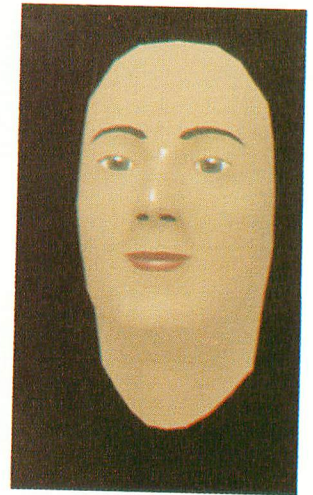
- Blinn, J. (1982) "A Generalization of Algebraic Surface Drawing" *ACM Transactions on Graphics*, 1, 235.
- Fournier, A., Fussell, D., and Carpenter, L. (June 1982) "Computer Rendering of Stochastic Models" *Commun. ACM*, 25, 6, 371-384.
- Hill, D.R., Witten, I.H., and Jassem, W. (1978a) "Some results from a preliminary study of British English speech rhythm." *Research Report 78/26/5, Dept. of CS, University of Calgary presented to the 94th. Meeting of the Acoust. Soc. Amer. Miami Dec 1977.*
- Hill, D.R. (1978b) "A program structure for event-based speech synthesis by rules within a flexible segmental framework" *Int. Journal of Man-Machine Studies* (1978), 10 (3) 285-294.
- Huitric, H. and Nahas, M. (March 1985) "B-Spline Surfaces: A tool for computer Painting" *IEEE Computer Graphics & Applications*, 5 (3) 39-47.
- Levitan, E.L. (1977) "Electronic Imaging Techniques" *Van Nostrand Reinhold Co., New York, N.Y.*
- Mandelbrot, B. (1983) "The Fractal Geometry of Nature." *W.H. Freeman and Company. (First Edition 1977).*
- O'Neill, J.J. and Oyer, H.J. (1961) "Visual Communication for the Hard of Hearing" *Prentice-Hall Inc., Englewood Cliffs, N.J.*
- Parke, F.I. (Dec. 1974) "A Parametric Model for Human Faces" *PhD. dissertation, University of Utah.*
- Parke, F.I. (Nov. 1982) "Parameterized Models for Facial Animation" *IEEE Computer Graphics and Applications*, 2 (9) 61-68.
- Platt, S.M. and Badler, N.I. (Aug. 1981) "Animating Facial Expressions" *ACM Computer Graphics (SIGGRAPH '81)*, 15 (3) 245-252.
- Reeves, W. (Apr 1983) "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects" *ACM Transactions on Graphics*, 2, 91-108.
- Walther, E.F. (1982) "Lipreading" *Nelson-Hall Inc., Chicago, Illinois.*
- Wyvill, B.L.M., McPheeters, C., and Garbutt, R. (July 1985a) "A Practical 3D Computer Animation System" *The BKSTS Journal (British Kinematograph Sound and Television Society)*, 67 (6) 328-332.
- Wyvill, B.L.M., McPheeters, C., and Novacek, M. (June 1985b) "Specifying Stochastic Objects in a Hierarchical Graphics System" *Proc. Graphics Interface 85, Montreal.*
- Wyvill, G., Wyvill, B., and McPheeters, C. (October 1985c) "Soft Objects" *Research Report No. 85/215/28, University of Calgary, Department of Computer Science.*



a)



b)



c)

Figure 1. Blink.

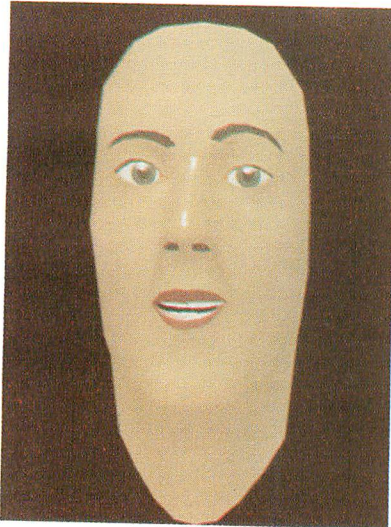


Figure 2. Smile.

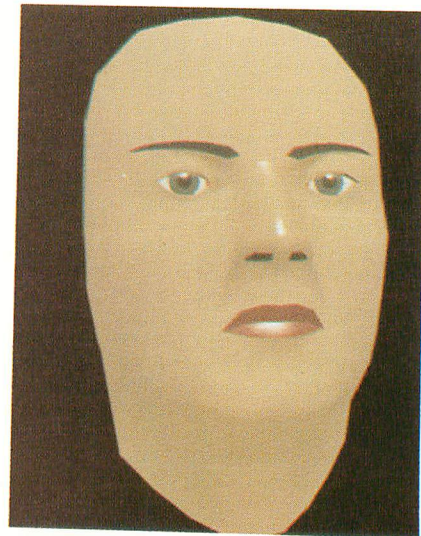


Figure 3. Frown.

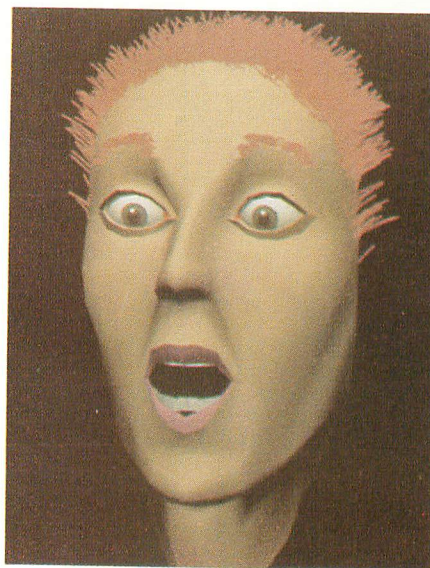
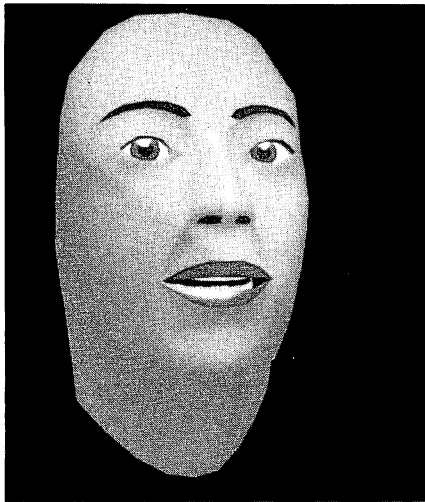
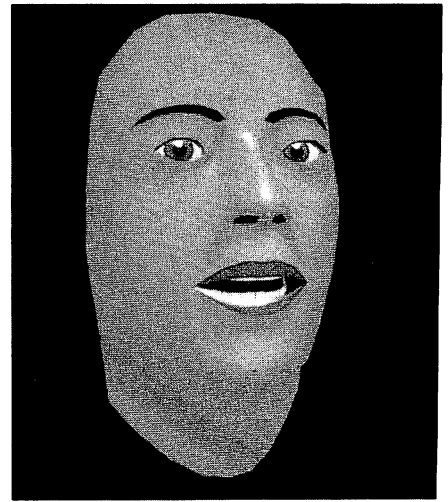


Figure 4. Scream.



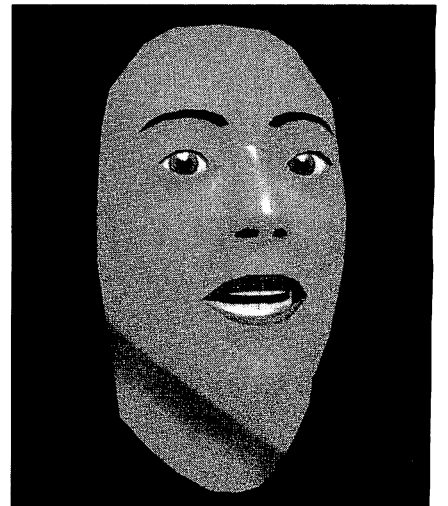
a) 'h'



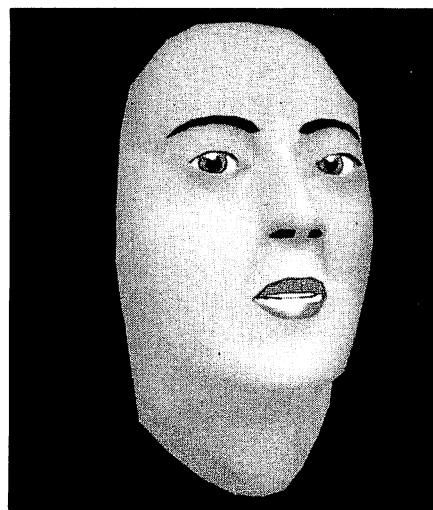
b) 'i'



c) 'th'



d) 'e'



e) 'r'

Figure 5. Hi there.

VIRYA - A MOTION CONTROL EDITOR FOR KINEMATIC AND DYNAMIC ANIMATION

Jane Wilhelms
Computer Graphics and Imaging Laboratory
Computer and Information Sciences Board
University of California, Santa Cruz, CA. 95064, U.S.A.

Abstract

Virya is an interactive graphical motion control editor for kinematic and dynamic animation. Most animation is controlled *kinematically*, by designating objects' positions taken over time without consideration for the causes of the motion. An alternative is *dynamic* motion control, where objects are seen as masses moving under the influence of forces and torques. Dynamic motion control has some advantages in that motion more naturally simulates real world conditions and many complex motions can be automatically calculated, though calculating motion is quite expensive and control is sometimes less intuitive. The editor *Virya* works both for kinematic and dynamic motion control. It has two main tasks: to specify *control functions* representing positions (kinematic) or forces and torques (dynamic) controlling motion, and to specify *control modes* which designate how control functions are interpreted or whether joints are frozen in place, relaxed, or balanced. Using these control modes, the user can designate motion using a convenient kinematic method and still use dynamic analysis as a final step to constrain and add realism.

KEYWORDS: computer animation, human modeling, dynamics, simulation

1. Kinematic Motion Control Methods

Motion control is a central problem in computer animation and is the one aspect of animation that most sets it off from other areas of computer graphics. Common kinematic approaches to motion control are *3-D keyframing*, *motion control functions*, *parametric control*, and *animation languages* (these approaches can, and often are, combined).

In 3-D keyframing, the user typically positions the objects in the scene interactively, designating a sequence of configurations and the times when they should occur [10,11]. The animation system then interpolates between these *key-frame* configurations to generate the *inbetween* configurations. 3-D keyframing is inherently superior to 2-D keyframing for 3-D animation because the problems of information loss do not occur. However, keyframing is limited by the necessity of creating many keyframes and the lack of complete control over the interpolation process defining the path and speed of motion between keyframes.

Parametric motion control involves designating certain parameters whose values define a particular configuration of the objects in the world [5]. For example, in the case of facial animation, parameters may designate the position of the mouth, the elevation of the eyebrows, etc. [8]. Parameters are convenient to use and allow association of reasonably complex motions (such as smiling) with one or a few parameters. Choosing parameters that cover the desired range of motion can be problematic, so the user may have to sacrifice complete motion control for ease of use.

Animation languages are an attractive alternative because complex motion can be described in the form of scripts [9]. Some languages are fairly low-level and merely provide a convenient interface to specify simple motions [7]. Higher-level languages would allow the user to specify motion in general terms (e.g. "walk forward") and depend upon an intelligent hierarchical interpretation system to find the specific low-level directions needed to draw the frames [16]. While high-level languages may provide the most convenience to the user in the long run, at present many issues involved in high-level motion control remain unresolved. Again, use of a script may limit the amount of control the user has over the motion.

Kinematic motion control functions represent motion at each degree of freedom in the form of position versus time curves. The control functions can be simply generated and succinctly stored using control points which generate the curve. These control functions are low-level and represent motion at individual degrees of freedom, but do allow very detailed specification of motion. An advantageous feature of control functions is that the final motion description of the other methods (changes to particular degrees of freedom over time) can be easily represented in this form. The use of an interactive control function editor allows the user to make individual changes to motion at the lowest-level and at the last minute, and can make up for some of the loss of exact control often concomitant with the above methods.

2. Dynamic Animation

Most animation systems at present are *kinematically-based*, that is, motion is considered as the relation of position versus time without consideration of the environmental influences causing the motion. *Virya*, the motion control editor described here, was designed mainly for use with the *dynamic* animation system *Deva*. In *Deva*, objects are considered as extended masses which act under the influence of forces and

torques. Using dynamic analysis, this relationship is formulated as the dynamics equations of motion; the solution of these equations is a kinematic description of the motion that would occur under the specified conditions in a "real" (simulated) world [13,14].

Dynamic animation has certain advantages lacking in kinematic systems. Motion is automatically constrained to respond to environmental conditions. For example, it is kinematically difficult to animate a body, such as a human figure, naturally responding to collisions, such as hitting the ground. One problem is that colliding objects should not move through each other. The user can avoid this, at considerable expense, by visually checking for unrealistic intersections, doing automatic collision detection, or implementing constraints using inverse kinematics [4]. Another problem is that when an articulated body collides, the motion of its many connected segments can be extremely complex and difficult to predict. Using dynamics, such collisions can be automatically simulated by applying an opposing force against the body when it collides. Taking this force into account during dynamic analysis results in a natural response to the collision, including a certain amount of bouncing and correct reaction of other connected body parts. Bodies will also automatically respond to gravity or the motion of connected body parts or external influences.

Dynamic animation does have some disadvantages as well. It is computationally much more expensive than kinematic animation [1,14]. When the system dynamics are complex, as in the case of articulated bodies with many degrees of freedom, numerical instability can be a problem. Dynamics also requires initial determination of object and environmental characteristics such as masses, joint limits, scale factors for springs and dampers used in collisions, etc. Perhaps the most serious disadvantage occurs in simulating controlled motion. While bodies respond naturally to environmental conditions, it is difficult to find when, where, and how strongly to apply internal forces and torques simulating muscles in humans and other animals. These issues are explored in detail elsewhere [13,14,15]. Suffice it to say that while dynamic animation is still in its exploratory stages, it does offer a method with considerable potential for simulating realistic motion.

3. Virya Motion Control Functions and Modes

Virya is a motion control editor used to develop, modify, and store motion control information in the form of *control functions* and *control modes*. *Virya* was designed to work within the dynamic animation system, *Deva*. Some of its features are only relevant to dynamic animation systems; other features are relevant to both kinematic and dynamic systems. *Deva* and *Virya* were specifically designed to explore the problems of controlling the motion of articulated bodies such as animals and robots. Though the same principles apply to controlling other, simpler objects, discussions will largely be from the standpoint of controlling articulated bodies.

Virya control functions can either be kinematic, representing positions over time for each degree of freedom, or dynamic, representing forces (sliding joints) or torques (revolute joints) for each degree of freedom. In *Virya*, control functions are represented by cubic interpolatory spline curves, piecewise curves that interpolate a sequence of user-defined control points with first and second derivative continuity [2,3].

Cubic interpolatory splines have the advantage that they interpolate specified control points; they have the disadvantages of being global (changes in one control point affect to varying degrees the entire curve) and given to occasional wild behavior. A local interpolatory spline, such as that of Kochanek [6], or a local approximating spline that can approach control points arbitrarily closely such as the beta-spline [3], may be more desirable. Control functions are easily constructed in *Virya* by picking control points on the screen using a puck and tablet.

Each degree of freedom of the body (e.g., flexion of the elbow or rotation of the head) can exist in one of five control modes for dynamic animation: *relaxed*, *dynamic control*, *frozen*, *balanced*, and *hybrid K-D* modes. Each degree of freedom can alternate between modes during the animation. One of *Virya*'s functions is to specify modes and their durations for each degree of freedom. A sixth, *pure kinematic* mode exists which completely by-passes dynamic analysis. In this case, control functions represent positions over time and are directly sampled to produce purely kinematic animation.

3.1. Relaxed Mode

Dynamic animations can be developed without any user-specified motion at all, merely by placing the body in an unstable position and letting it react to the gravitational force, its own joint limits, the ground, etc. The degrees of freedom in *relaxed* mode will move freely under environmental conditions with no internal controlling force or torque simulating a muscle contraction or a robot actuator motor. These relaxed degrees of freedom are constrained to remain within their joint limits and their motion is slightly damped. Other forces or torques due to collisions or motion of other body parts will still act upon them. (Within the system *Deva*, springs and dampers are used to mimic both joint limits and collisions.)

3.2. Dynamic Mode

To actually control the animation, pseudo-muscular forces or torques must be applied to certain degrees of freedom of the body. For example, to make the body wave its arm, torques must be applied to the shoulder and elbow. The most direct way to specify these forces and torques is to develop a control function for each degree of freedom whose motion is controlled in this way. These control functions represent a force (for sliding degrees of freedom) or a torque (for revolute degrees of freedom) over time. The control functions are sampled to find the appropriate controlling force or torque to apply to specified degrees of freedom at each time instant that dynamic analysis is done; these controlling forces and torques are then added to the automatically calculated forces and torques mentioned above to find the total forces and torques acting upon the body.

While these force/torque control functions have the advantage of directly specifying the controlling forces and torques needed for dynamic analysis, they leave the user at the disadvantage of not knowing intuitively what forces or torques will be necessary to produce the desired motion. This problem is accentuated by the complex interactions between different parts of an articulated body. For example, should the user find the correct force to lift the arm rigidly at the shoulder, and then add torques to the elbow to accompany the lifting by a bending action, he will find that the additional torque at the elbow inter-

feres with the smooth motion of the shoulder. For this reason, other modes have been added for user ease.

3.3. Freeze Mode

It is common during many movements of articulated bodies that some parts of the body remain locally stable. For example, in reaching movements the legs and hips may remain stable, and during walking the head usually is directed forward. Because of the complex interplay mentioned above, it is difficult to find the sequence of forces or torques that will ensure local stability. A simple solution to this is to simulate the stabilizing torque (or force) with a tight spring and damper clamped about the local position. This can be viewed as a temporary change in the range of the joint end limits.

3.4. Balance Mode

While automatic response to gravitational forces make certain motions easy to simulate, this can create problems with coordinated movements. For example, walking no longer becomes a question of merely manipulating the legs in the proper sequence, but also of balancing the upper part of the body to keep it from falling over. A simple way to achieve balance is to describe a world-space orientation vector for particular segments, such as the trunk, and apply an external applied force to counteract any motion away from this orientation. Experimentation shows this technique is acceptable in limited cases; whether it will provide realistic balance in all cases has not been explored in depth.

3.5. Hybrid K-D Mode

The freeze and balance modes do help with some of the motion control problems introduced by dynamics, but leave the major problem of determining forces or torques for controlled motion, rather than just stability. An approach to this problem that has been reasonably successful is to describe the desired motion in kinematic terms, as kinematic control functions specifying rotation (for revolute joints) or translation (for sliding joints) over time. The exact same *Virya* interactive interface can be used; the interpretation of the control functions merely changes.

These kinematic descriptions are then used to find the forces and torques applied at specified degrees of freedom. The method used to find these forces and torques is trivially simple, but surprisingly effective. The equations used are

$$\begin{aligned}delp &= des_pos - pos \\delv &= delp/deltime - vel \\ft &= delv * m/deltime\end{aligned}$$

For sliding joints, *delp* is the difference between the desired position at the next time sample (*des_pos*) and the present position (*pos*). *delv* is *delp* divided by the time between samples (*deltime*) minus the present velocity (*vel*), in other words, the amount the velocity must be altered to achieve the desired position at the next time sample. *ft* is the estimated force that must be applied to achieve this, and *m* is the mass of the segment distal to this degree of freedom. For revolute joints, the same formula is used but the velocities are angular, *ft* is a torque, and *m* is a moment of inertia. Probably because

dynamic analysis is done from 3 to 30 times as frequently as images are displayed, no feedback is needed to achieve smooth motion.

The advantage to this method is that the user can enter motion directions in the intuitive kinematic form for those joints whose particular motion is known (or desired) and yet retain the advantage of dynamics.

3.6. Pure Kinematic Mode

Kinematic mode is separate from the dynamic animation package. *Deva* can operate as a strictly kinematic animation system, in which case the control functions specified by *Virya* are taken to represent the actual desired motion for all degrees of freedom. The dynamic analysis routines are entirely bypassed and fast kinematic animation is possible.

4. Virya User Interface

The *Virya* screen (see Figure 1) consists of three regions. The lower half of the screen contains small joint windows representing each degree of freedom of the system. The upper right quadrant consists of a menu of commands for designing and saving control functions. The upper left quadrant is a large window where control functions can be viewed and altered. The user selects menu items or designates points on the screen by using a graphics tablet and puck. *Virya* runs on an Evans and Sutherland PS300/340 graphics system.

Each joint window contains a label identifying the degree of freedom represented there; e.g. "J4 elbow (1) rz" refers to a z-rotation of the elbow (joint number 1). The line through each window joins the control points that define the curve representing motion control information for that degree of freedom. The user can alter these control points and thus define the shape of the motion control function. By specifying more or fewer points and spacing them differently, such control functions can be used to control both the path of the motion and its velocity (kinematics) or the strength of the force or torque applied (dynamics).

The menu consists of I/O commands, joint commands, vertex commands, and miscellany. The I/O commands are LOADBODY (input a body description including segments, joints, degrees of freedom, and present configuration); LOAD (input a previously created *Virya* file containing motion control information); and SAVE (save the present motion control description in a file). Joint commands are SHOWJ (bring a particular control function into the large window); COPYJ (copy the control function for one degree of freedom to another); ACTIVEJ (designate one control function as modifiable); REDRAWJ (redraw the active control function); and CURVEJ (use the control points to create a cubic interpolatory spline curve). The vertex commands all refer to the degree of freedom that has been designated "active". They are INITV, ADDV, DELV, MOVEV, and LOCV. INITV initializes the control function to a horizontal line along the time axis. ADDV, DELV, and MOVEV add, delete and move control points defining the control curve. LOCV gives the exact numeric value of a point on the screen. The miscellaneous commands are CONFIRM (confirm a change); QUIT (leave *Virya*); and TABLET (a binary switch between inputting control points from the terminal or from the graphics tablet).

The DKFRB (*dynamic / kinematic / frozen / relaxed / balanced*) option allows the user to designate control modes for each degree of freedom and a time span during which the modes are in force. The default control mode for dynamics is dynamic mode.

Virya motion descriptions are stored in an ascii file (which under present conditions makes up for space consumption with user convenience); part of a sample file is shown in Figure 3.

5. Use of Virya with 3-D Keyframing

Because motion of a complex articulated body such as a human figure is sometimes difficult to visualize in terms of angular or translational motion of individual degrees of freedom, it has been found convenient to initially define the motion as a series of keyframes developed by interactively positioning the body on the screen using *Deva*. A sequence of keyframes with the times they should occur are stored in an ascii file (part of such a file is shown in Figure 2). These files can be converted to *Virya* control function files and used to generate control curves which are sampled to drive the animation or called into the *Virya* editor for modification.

These key-frame-derived control functions initially place all degrees of freedom in hybrid K-D mode, but because this almost completely constrains the motion, there would be little point in doing dynamics. Typically the user modifies this file placing many degrees of freedom into relaxed, frozen, or balanced modes to allow dynamics to fulfill its purpose of adding realism.

6. Sample Session

A simple session using *Virya* will be described to illustrate its use. First, the user enters the animation system *Deva* and calls up a previously stored figure, in this case the 24-degree-of-freedom human figure *Joe*. Using dials and the keyboard, five keyframe configurations for *Joe* are found. These configurations are to occur at 0, 3, 6, 7, and 8 seconds in the animation. They are stored in ascii form in a keyframe file shown in part in Figure 2. The lines with a single number represent the times when the keyframes occur, and the following numbers are positions for each degree of freedom of motion at that time.

The keyframe file is converted using *Deva* to the format needed for interaction with *Virya*. In the keyframe file, positions are grouped by the times when they occur; in the *Virya* file they are grouped by degree of freedom. Figure 3 shows part of the *Virya* file originally derived from the keyframe file, but somewhat modified using *Virya*. Following the control positions (taken from the keyframes) for each degree of freedom are definitions of the states, or modes, over time. Initially, when the file is created from a keyframe file, all degrees of freedom are in the hybrid K-D mode (K) during the default time span (0-100 seconds). (The 4 numbers after the time span are only used in balance mode, where they represent the position vector and the amount of deviation from it allowed.) During the motion described (lifting the legs and swinging the arms), many degrees of freedom (such as the waist) are frozen into their local configuration and others (such as the right knee) are relaxed. The modes of these joints are changed from the default K-D mode.

This *Virya* file was used to drive dynamic analysis and the resultant, predicted motion stored in another, similar *Virya* file. This second, output file is a kinematic description of the dynamically predicted motion, and was sampled to produce the animation shown in Figure 4. Keyframes are approximately in locations (0,0), (1,3), (3,0), (3,2), and (3,5) (row major order). (Actual dynamic analysis was done 300 times per second; not all of these configurations are stored for the kinematic description and display.)

7. Conclusions

The interactive graphical editor *Virya* is used to design and store motion control commands for kinematic or dynamic animation using control functions and control modes. For kinematic animation, the user designs control functions representing positions over time for each degree of freedom. For dynamic animation, control functions may represent either kinematic information (positions over time) or dynamic information (forces or torques over time). To alleviate some of the control problems that accompany the advantages of dynamic animation, the freeze, balance, and relaxed control modes are also available. The kinematic output of dynamic analysis routines and motion control information derived from other higher-level control methods can be stored the form of *Virya* data files. This format is convenient for low-level modification and sampling for animation generation.

References

1. William W. Armstrong and Mark Green, "The Dynamics of Articulated Rigid Bodies for Purposes of Animation," pp. 407-415 in *Proceedings of Graphics Interface '85*, Computer Graphics Society, Montreal (May, 1985).
2. Brian A. Barsky and Spencer W. Thomas, "TRANSPLINE - A System for Representing Curves Using Transformations among Four Spline Formulations," *The Computer Journal*, Vol. 24, No. 3, August, 1981, pp. 271-277.
3. Richard H. Bartels, John C. Beatty, Brian A. Barsky, *An Introduction to the Use of Splines in Computer Graphics*, Technical Report No. UCB/CSD 83/126, Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, California, USA (August, 1983).
4. Michael Girard and A. A. Maciejewski, "Computational Modeling for Computer Generation of Legged Figures," pp. 263-270 in *Proceedings of ACM SIGGRAPH '85*, 19, San Francisco, Ca. (July, 1985).
5. Patrick Hanrahan and David Sturman, "Interactive Animation of Parametric Models," pg. 102-111 in course notes for the tutorial in *Introduction to Computer Animation*, ACM SIGGRAPH '85.
6. Doris H. U. Kochanek, Richard H. Bartels, and Kellogg S. Booth, *A Computer System for Smooth Keyframe Animation*, University of Waterloo, Waterloo, Ontario (December 1982).
7. T. J. O'Donnell and Arthur J. Olson, "GRAMPS - A Graphical Interpreter for Real-Time Interactive Three-Dimensional Picture Editing and Animation," pg. 133-142 in *Proceedings of ACM SIGGRAPH '81*, 15, Dallas, TX (July 1981).
8. Frederic Parke, "Parameterized Models for Facial Expression," *IEEE Computer Graphics and Applications*, 2, No. 1 (November 1982), pg. 61-68.

Figure 1. The *Virya* Screen

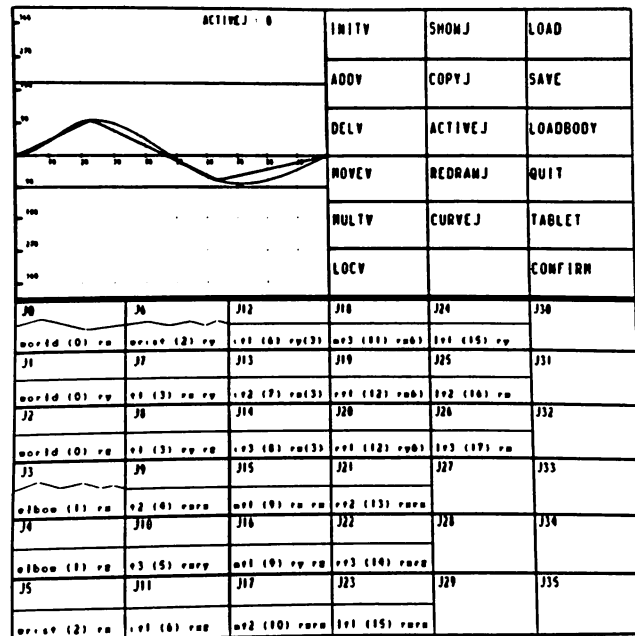


Figure 2. Partial Keyframe File

```
0.000000
180.000000 180.000000 180.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 -180.000000 0.000000
0.000000 0.000000 -180.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000

3.000000
180.000000 180.000000 180.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 -150.000000 0.000000
30.000000 0.000000 -210.000000 0.000000
25.000000 0.000000 30.000000 0.000000
0.000000 0.000000 0.000000 0.000000

6.000000
180.000000 180.000000 180.000000 0.000000
0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 -210.000000 0.000000
30.000000 0.000000 -180.000000 0.000000
25.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000
```

Figure 3. Partial *Virya* File

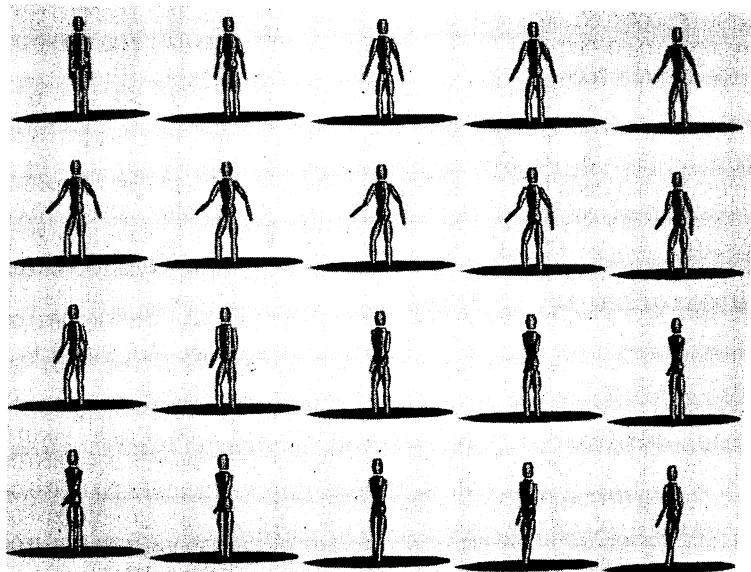
```
dof 18 Pjnt 7 Type 0 Numv 5 j_hipr
Control 0.000000 0.000000
Control 3.000000 0.523599
Control 6.000000 0.000000
Control 7.000000 0.000000
Control 8.000000 0.000000
States K 0 6 0 0 0
States F 6 8 0 0 0
Q

dof 19 Pjnt 7 Type 1 Numv 5 j_hipr
Control 0.000000 0.000000
Control 3.000000 0.000000
Control 6.000000 0.000000
Control 7.000000 0.000000
Control 8.000000 0.000000
States F 0 8 0 0 0
Q

dof 20 Pjnt 8 Type 0 Numv 5 j_kneer
Control 0.000000 0.000000
Control 3.000000 0.000000
Control 6.000000 0.000000
Control 7.000000 0.000000
Control 8.000000 0.000000
States R 0 8 0 0 0
Q
```

9. Craig Reynolds, "Computer Animation with Scripts and Actors," in *Proceedings of ACM SIGGRAPH '81*, 15, Dallas, TX (July 1981).
10. Craig Reynolds, "Description and Control of Time and Dynamics in Computer Animation," pg. 289-296 in course notes for the tutorial in *Advanced Computer Animation*, ACM SIGGRAPH '85.
11. Scott Steketee and Norman I. Badler, "Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control," in *Proceedings of ACM SIGGRAPH '85*, 19, San Francisco, CA (July, 1985).
12. David Sturman, "Interactive Keyframe Animation of 3-D Articulated Models," pg. 102-111 in course notes for the tutorial in *Introduction to Computer Animation*, ACM SIGGRAPH '85.
13. Jane Wilhelms and Brian A. Barsky, "Using Dynamic Analysis for the Animation of Articulated Bodies such as Humans and Robots," pp. 97-104 in *Proceedings of Graphics Interface '85*, Montreal (27-31 May 1985).
14. Jane Wilhelms, *Graphical Simulation of the Motion of Articulated Bodies such as Humans and Robots, with Special Emphasis on the Use of Dynamic Analysis*, Electrical Engineering and Computer Sciences, University of California, Berkeley, CA (July, 1985).
15. Jane Wilhelms, "Using Dynamic Analysis for Realistic Animated Motion," submitted for publication 1986.
16. David Zeltzer, "Motor Control Techniques for Figure Animation," *IEEE Computer Graphics and Applications*, 2, No. 9 (November 1982), pg. 53-59.

Figure 4. Dynamic Animation Based on Virya File



NEAR-REAL-TIME CONTROL OF HUMAN FIGURE MODELS

W.W. Armstrong, M. Green, and R. Lake
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada

ABSTRACT

The animation of human figures is one of the major problems in computer animation. A recent approach to this problem is the use of dynamic analysis to compute the movement of a human figure given the forces and torques operating on the body. One of the main problems with this technique is computing the forces and torques required for particular motions. As a solution to this problem an interactive interface to our dynamics routines has been produced. This interface, along with a collection of low level motion processes, can be used to control the motion of a human figure model. In this paper both the user interface to our dynamics routines and the motion processes that we use are described.

KEYWORDS: human figure animation, dynamic analysis, interactive control of human figures

1. Introduction

One of the more challenging parts of computer animation is the animation of human figures and other articulated bodies (for example, robots and animals). Over the past decade a number of techniques have been developed for the animation of human figures. These techniques vary from digitizing human movement, to the development of kinematic models of human motion. Recently the dynamic analysis of human motion has been proposed as a way of animating human figures [Armstrong and Green 1985a, Wilhelms and Barsky 1985].

Dynamic analysis has a number of advantages over other approaches to human animation. Since this technique is based on well known techniques from physics and robotics, it is capable of producing very realistic motion. The motion of the human figure is controlled by forces and torques that are applied to the limbs of the body. In most motions only a small number of the limbs are actively involved, these limbs are called the controlled limbs. The other limbs in the body either maintain the same relative position, or follow the motion of the controlled limbs. The latter motion can be automatically produced by the dynamics software, therefore, the animator only needs to specify motion information for the controlled limbs. The small volume of information required to produce motion could lead to human animation systems that are much easier to use than existing systems.

There are two main problems associated with the use of dynamic analysis for human animation. The first problem is the amount of computer time required to compute the motion

of the human figure. Traditional approaches to the computation of human motion require overnight batch runs for simple animation sequences [Wilhelms 1985]. We have developed an approach to the solution of the equations of motion that is significantly faster than other techniques. This approach can produce near-real-time animation on commonly available hardware. Some of our results in this area are described in section 2.

The second problem is determining the torques and forces required to produce a particular motion sequence. Animators work in terms of body positions and complex motions, such as walking and running. They have no experience with the torques and forces required to produce the motion they want. There are two parts to our solution to this problem. The first part is the development of a number of low level motion processes. These processes generate the torques and forces required to produce particular types of motion. The second part of the solution is an interactive user interface that allows the animator to specify values for the parameters used by the motion processes, or directly apply torques and forces to the body while it is in motion. The animator can obtain immediate feedback on the effects of changes in parameter values, or the effects of torques and forces. Through this interface the animator is able to experiment with different ways of producing motion, and develop a feel for how they can be used to produce the motion he or she wants. There is also the possibility of producing canned motions that can be called upon by the animator. These motions could be parameterized so they can be customized to a particular situation. The work we have done in this area is discussed in sections 3 and 4.

2. Near-Real-Time Dynamics

One of the main drawbacks to using dynamic analysis for human animation has been the amount of computing required. Some of the formulations of the equations of motion for human figures and techniques for their solution are based on the techniques developed in mechanical engineering for the analysis of general linkages such as those found in machines. The linkage structure of the human body is not as complicated as the systems studied in mechanical engineering, where any of the links in the mechanism could directly effect the motion of any of the other links. This can give rise to a graph structure for the links. On the other hand, the human body can be viewed as a tree of links with no interconnections between the leaves on different branches. This observation significantly simplifies the equations of motion and allows for efficient solution techniques. This version of the equations of motion and techniques for their solution have been described elsewhere [Armstrong and Green 1985a,b]. At this point we will summarize the results of this work.

Two implementations of our solution of the equations of motion have been produced. The first implementation is designed to run on a single processor. This implementation is written in C and currently runs on a DEC VAX 11/780, SUN workstation, and IRIS workstation. The time required to compute a motion sequence depends upon the inertias used for the body parts. The step size required for a stable solution of the equations of motion is proportional to the square root of the values of the inertias. In our current implementation the inertias are about a factor of 10 larger than those found in a human body. This results in a computation time that is within a factor of 3 to 10 of real-time depending on the complexity of the figure and the complexity of the motion. The animation produced by this implementation is fast enough to get a good feel for the motion while the program is running. The other implementation is designed to run on a network of processors [Armstrong et.al. 1986]. Since the human body can be viewed as a tree of limbs, subsets of limbs of the tree can be assigned to different processors, and a large amount of the computation can proceed in parallel. The results we have obtained so far indicate that real-time animation could be produced by a network of four SUN 3 workstations.

These results indicate that it is possible to construct a system where the animator can manipulate the dynamics of a human figure in real-time.

3. Control Strategies

Most human motion involves only a subset of the limbs in the body. When the animator develops these motions he or she will want to work with a small number of limbs at any point in time. Controlling more than two or three limbs in real-time is probably beyond the capabilities of most people. Thus, the animation system should allow the animator to build up his or her motion sequence on a limb-at-a-time basis. The main problem with this approach is that when one limb moves, the other limbs it is connected to are subjected to forces and torques as a result of its motion. This is a natural result of Newton's laws of motion. Some of these secondary motions may be desirable. For example, when the upper arm moves the animator will want the lower arm and all the limbs attached to it to also move. Other secondary motions are not desirable. A good example of this is when the figure reaches for an object only the arm should move and not the body as a whole.

The solution to this problem can be based on the use of a number of motion processes, which when added to the body model guarantee reasonable behavior. These motion processes are similar in function to the finite state automata used by Zeltzer [Zeltzer 1982]. A number of features of human motion can be handled by a collection of motion processes. Some of these features are: maintaining the same relative position between two connected limbs, balance, ground reaction, and the performance of simple motions, such as reaches. The motion processes can be divided into two basic categories; called limb processes and global processes. A limb process is responsible for the motion of only one limb. Each limb can have one or more motion processes associated with it at any one time. The number and types of motion processes on each limb are under the control of the animator. The global motion processes affect more than one limb. These processes are responsible for motions that require global knowledge of the state of the body. Examples of this type of process are balance and ground reaction.

The motion processes described in the following sections were motivated by the work that has been done in biomechanics (see [McMahon 1984a] for a good introduction to some of the relevant work). We have used biomechanics as a source of ideas for controlling the motion of the body, we are not trying to accurately model the human nervous or muscle systems.

3.1. Limb Motion Processes

The human figure model consists of a number of links, with each link representing a part of the body. The current model contains 14 links (head, neck, upper body, lower body, upper arm, lower arm, upper leg, lower leg, and foot). At the proximal end of each link there is a three-degree-of-freedom rotational joint connecting it to its parent. The upper body link has three extra degrees of freedom representing the translation of the body with respect to the world coordinate system. The current state of the model is given by the position of the upper body and the three rotation angles at each joint.

There are nine parameters that can be used to control the motion of each link. These parameters are the components of the the internal torques (torques generated at the joints), external torques (torques applied from outside of the body), and external forces (forces applied from outside of the body). Only the internal torques are used by the limb motion processes. At any point in time there can be one or more motion processes associated with each limb. Each of these processes contributes to the internal torque that is applied to that limb.

The motion processes are controlled by a joint information table. This table contains one entry for each degree of freedom in each joint. The table entry contains the state of the degree of freedom and parameter values that are required by the associated motion processes. The state is a bit vector that specifies the motion processes that are currently associated with that degree of freedom. There is one bit in this vector for each motion processes. If the bit is set, the motion process can affect that degree of freedom.

The motion processes are executed on each iteration of the dynamics calculations. Each degree of freedom in each limb is considered separately. At start of the processing for a degree of freedom, its internal torque is set to zero. Then the state bit vector is examined to determine the motion processes that are to be executed. Each motion process uses the current state of the link, plus its own parameters (stored in the joint information table) to compute a contribution to the internal torque. At the end of this process, new internal torques have been generated for each limb in the model.

At the present time we are using four motion processes. The first process, called free swing, is a null process that does not contribute to the internal torque of the joint. This process allows the joint to move freely (without any constraints) in the degree of freedom it is attached to.

The second motion process, called friction, generates a velocity dependent friction which is used to slow the limb down when it is in motion. The friction in a joint is proportional to the relative angular velocity of the limb with respect to its parent. The constant of proportionality can be interactively controlled by the animator. This model of friction agrees with results from biomechanics [McMahon 1984a].

The third motion process, called the maintain process, is used to maintain the relative angular positions between two adjacent limbs. When producing a motion, the animator may want only a subset of the limbs to move. The other limbs in the body should stay in the same relative positions. This motion process is used to achieve this goal. When using this motion process the animator specifies a parameter, called center, which is the desired angle between the limb and its parent. The motion process maintains the angle between the two limbs close to the value of center. The angle between limbs cannot be clamped to the center value for two reasons. First, this behavior is not realistic from a biological point of view. Second, fixing an angle adds a constraint to the dynam-

ics equations and would require reformulating the solution. The following function, $T(x)$, is used to determine the torque applied to a joint given the angle, x , at the joint, and the desired angle, center.

$$T(x) = \alpha (\exp(\beta(x - \text{center})) - 1) \quad \text{if } x \geq \text{center} \quad (1)$$

$$= -\alpha (\exp(\beta(\text{center} - x)) - 1), \quad \text{otherwise}$$

The parameters α and β , which can be set by the animator, determine the strength of the torque applied at the joint. Initial values are supplied for center, α , and β that will maintain the human figure in a standing position. The form of this motion process is based on results from studies on muscle dynamics [McMahon 1984a] [Hatze 1977].

The fourth motion process, called the simple move process, is used to move a limb from one position to another. The animator specifies the new angle between the limb and its parent, and this motion process produces a smooth motion between the current limb position and the new limb position. When the new limb position is reached the maintain process is invoked to maintain the new position. In order to move the limb from its old position to its new position a sequence of torques (one for each time step of the dynamics calculations) must be applied to the joint. This sequence of torques must satisfy two conditions. The first condition is that the torques must be strong enough to move the limb to its new position. The second condition is that the generated motion must appear to be natural.

In order to satisfy the first condition we use expression (1) to estimate the amount of torque required to reach the new position (the new position is used as the value of center). If this torque was applied to the joint, the limb would reach its new position within one or two iterations of the calculations. Time steps on the order of 0.01 seconds are usually used in the dynamics calculations, therefore, the limb would move from its original position to its new position in a small fraction of a second. For most types of motion this change of position is far too rapid. The torques produced by expression (1) are unrealistic, but they do guarantee that the limb will reach the new position, thus it is a good starting point for our calculations.

In order to produce more realistic motion we place two constraints on the torque produced by expression (1). The first constraint is that the torque cannot exceed a certain maximum value (these values can be found in tables of maximum torques for physical activities [Plagenhoef 1971]). The second constraint is that the rate of change of torque cannot exceed a maximum value (reasonable values for this parameter are scattered in the literature). When these two constraints are applied to the torques produced by expression (1), smooth motion is produced in the first part of the action. The main problem with this technique is that the motion does not slow down as the final position is reached. Even with this problem the results look fairly realistic.

The motion in the second half of the action can be improved by decreasing the torque applied to the limb as it approaches the new position. The angular mid-point of the motion is fairly easy to determine (the average of the initial and final angles). After this point the torque applied to the limb should be decreasing. This can be achieved by setting the maximum torque to the torque value at the mid-point of the action. At each iteration after the mid-point, the maximum torque is decreased by the square root of the ratio of the distance from the current position to the goal, to the distance from the center position to the goal. That is, we use the following expression:

$$\text{max_t} = \text{center_t} \quad ((\text{current} - \text{goal}) / (\text{half} - \text{goal})) \quad (2)$$

where: center_t = the torque at the angular mid-point of the action
current = current joint angle
goal = final joint angle
half = the mid-point of the action

The above expression seems to produce a smooth motion in the second half of the action.

3.2. Global Motion Processes

The global motion processes use the states of several limbs in order to control the global motion of the body. These processes can generate torques and forces that are applied to several of the body's limbs. At the present time two global processes are used in our software. These processes maintain the balance of the figure, and its reaction to the ground.

At the present time, a very simple approach is taken to balancing the figure. First, the difference between the positions of the top and bottom of the body is calculated. A restorative force based on this difference is then applied to one or more limbs of the body, depending upon the type of motion. This technique can be used to keep the body in a standing position, but in general it is too restrictive. A more realistic balancing technique would be based on the limbs that are in contact with the ground. Torques generated by these limbs could be used to keep the body in balance.

One of the main problems with balance is that different types of balance may be required for different types of motion. In the case of diving and gymnastics, a balance process may hinder the motion. Walking is based on falling forward [McMahon 1984b], so an external balance process could make it impossible for the figure to walk. In other motions, such as reaching and lifting, balance is very important, so for these motions a balance process must be used. This suggests that either a sophisticated model of balance must be developed, or it must be under the control of the animator.

The human figure must be able to react to any of the objects it comes in contact with. The most common of these objects is the floor or ground. In equilibrium the floor will exert a force on the body equal to the body's weight. This solution cannot be used in animation, since the motion of the body will change the force applied to the floor by the body. The approach that we have used is to monitor the position of the body's feet (or another points of contact with the floor). A spring force is applied to the feet in order to keep the feet at the floor level. This technique works as long as the feet stay close to the floor. If the body is falling towards the floor, a more sophisticated technique is required. The friction between the feet and the floor must also be considered, otherwise the feet will slide all over the floor. A good discussion of ground reaction can be found in [Wilhelms 1985].

4. Software Architecture

The interactive animation system that we have developed is divided into two main components, which are shown in fig. 1. The first component, called the front end, is responsible for displaying the human figure model and interacting with the user. This component of the animation system resides on an IRIS 1400 workstation and is responsible for controlling the animation system. The second component is the dynamic analysis program. This program performs all the dynamics calculations for the human figure model. The second component can reside on the IRIS workstation, or on one or more of the other processors on our local network.

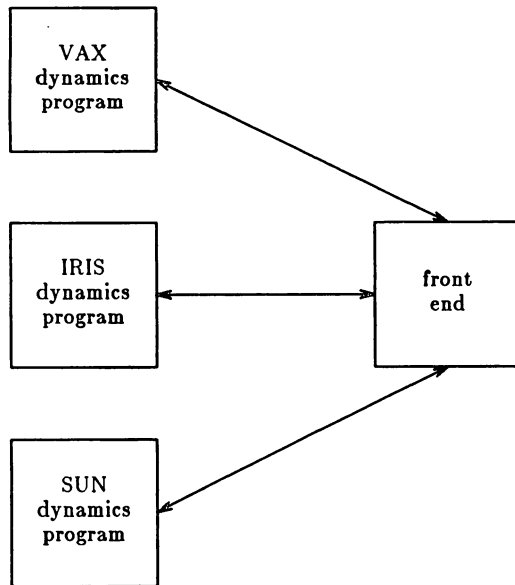


Fig. 1 Software Architecture

The two components of the animation system communicate by sending packets over an interprocess communications facility (either a pipe, or a socket in the case of an ethernet connection). The front end invokes the back end process when the user wants to perform dynamics computations. Upon invocation, the dynamics program reads a start-up file containing a number of parameters for the computation, and performs the first step in the computation. At the end of the first step the dynamics program sends a set of packets to the front end. There is one packet in this set for each limb in the body, giving its current joint angles. There is also a packet specifying the current position of the root limb within the world coordinate system. At this point the front end program responds with one or more packets. These packets are used to change the state of a limb, pass parameters for the motion processes, or specify a torque or force to be applied to the body. The last packet in this exchange is a Next_step packet sent from the front end to the dynamics program. At this point the dynamics program starts the next calculation cycle. This packet exchange ensures that the front end and the dynamics program are always in step.

A packet exchange need not occur at each time step of the computation. The time step used in the computation is of the order of 0.01 seconds. This time step is too fine for display and the types of control we are using. Currently, the packet exchanges occur every 0.05 seconds of simulation time. This rate is sufficient for both display and control.

This division of the animation system into separate processes has two main advantages. First, separating the dynamics computations from the display and user interface allows the use of either the single processor or distributed versions of the computations with the same user interface. In other words, as far as the animator is concerned interacting with the single processor and distributed version of the computations is the same. The only noticeable difference is the speed of computation. This allows the animator to take advantage of the available computing resources without changing his mode of operation. Second, the use of separate processes allows several people to work on the project without interfering with each

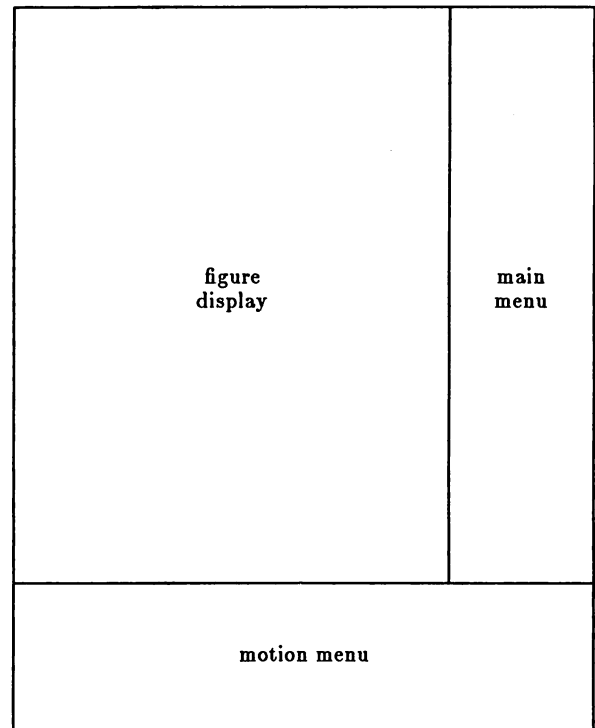


Fig. 2 Screen Layout

other.

The screen layout for the front end is shown in fig. 2. The display screen is divided into three main sections, called the figure display, main menu, and motion menu. The figure display is used for displaying a graphical representation of the human body. The human body is represented by a three dimensional polygon model, with four to six polygons defining each of the limbs. Each of the polygons in a limb has a different colour facilitating the identification of the different sides of the figure. This type of model allows us to display the human figure at a rate of 14 frames per second.

The main menu contains a collection of commands that can be invoked when the dynamics computations are not being performed. The commands on this menu are used to change the eye position, record the motion produced by the dynamics routines, start the dynamics computations, playback a motion sequence that has been previously computed, save a motion sequence on a disk file, and retrieve a previously computed motion sequence from disk.

Once the dynamics computations have been started they can be interrupted in two ways. If the user presses one of the mouse buttons, the dynamics computations are suspended at the next packet exchange, and control is transferred to the main menu. The dynamics computations can be restarted by selecting the dynamics command from the main menu. This facility allows the user to change the eye position, or record parts of the motion while the computations are in progress.

The other way of interrupting the dynamics computations is to move the mouse into the motion menu area. At this point the cursor changes shape, and the user can select one or more commands from the motion menu. The motion menu contains the name of each joint in the body and the name of the parameters for the motion processes. In order to change a parameter value for a motion process, the user selects the joint and parameter name from the menu, and then selects the value command. At this point the user is prompted for the new

value for the parameter. Similarly the user can change the state of any of the limbs in the body. When the user is finished modifying the motion processes, he or she can move the mouse into the figure display in order to resume the computation.

This user interface has three main advantages over batch dynamics computations. First, at any point in the computation the user can suspend the computation, and then playback (in real-time) the motion sequence that has been produced. This allows the user to terminate computations that are not producing the desired motion before the end of the motion sequence. This saves both animator and machine time. Second, the animator can interactively change the motion as it is being computed. This allows the animator to react to the motion of the human figure, and frees him from precisely timing the movements of the figure. Third, the near-real-time computation of the motion allows the animator to experiment with different types of control strategies.

5. Summary

In this paper we have reviewed some of the work that has been done on applying dynamic analysis to the animation of human figures. We have also summarized the work we have done on producing efficient algorithms for solving the equations of motion and their implementation in both a single processor and multiple processor environments.

The significant new material in this paper is the discussion of automatic motion processes for controlling the human figure and the interactive system that we have developed for human figure animation. This interactive animation system allows the animator to take advantage of the power and flexibility of the new dynamic analysis techniques.

Acknowledgements

The work reported here was supported by the Natural Science and Engineering Research Council of Canada and the University of Alberta. We would also like to thank our support staff, in particular S. Sutphen and M. Olafsson, for their assistance.

References

- [Armstrong and Green 1985a] Armstrong W.W., M.Green, "The Dynamics of Articulated Rigid Bodies for the Purposes of Animation", Graphics Interface'85, p.407-415, 1985.
- [Armstrong and Green 1985b] Armstrong W.W., M.Green, "The Dynamics of Articulated Rigid Bodies for the Purposes of Animation", The Visual Computer, vol.1, no.4, 1985.
- [Armstrong et.al. 1986] Armstrong W.W., T.A. Marsland, M. Olafsson, J. Schaeffer, "Solving Equations of Motion on a Virtual Tree Machine", Technical Report, Department of Computing Science, University of Alberta, 1986.
- [Hatze 1977] Hatze H., "A Myocybernetic Control Model of Skeletal Muscle", Biological Cybernetics, vol. 25, p.103-119, 1977.
- [McMahon 1984a] McMahon T.A., *Muscles, Reflexes, and Locomotion*, Princeton University Press, Princeton N.J., 1984.
- [McMahon 1984b] McMahon T.A., "Mechanics of Locomotion", International Journal of Robotics Research, vol.3, no.2, p.4-28, 1984.
- [Plagenhoef 1971] Plagenhoef S., *Patterns of Human Motion: a cinematographic analysis*, Prentice-Hall, Englewood Cliffs NJ, 1971.
- [Wilhelms 1985] Wilhelms J.P., *Graphical Simulation of the Motion of Articulated Bodies Such as Humans and Robots, with Particular Emphasis on the Use of Dynamic Analysis*, PhD Thesis, University of California, Berkeley, 1985.
- [Wilhelms and Barsky 1985] Wilhelms J., B. Barsky, "Using Dynamic Analysis to Animate Articulated Bodies such as Humans and Robots", Graphics Interface'85, p.97-104, 1985.
- [Zeltzer 1982] Zeltzer D., "Motor Control Techniques for Figure Animation", IEEE Computer Graphics and Applications, vol.2, no.9, p.53-59, 1982.

Modeling and Animating Three-Dimensional Articulate Figures

Danny G. Cachola
Gunther F. Schrack

The University of British Columbia
Vancouver, B.C.

ABSTRACT

This paper describes a method for representing and animating three-dimensional articulate figures. It permits the definition of a model consisting of segments and joints, and the specification of the model's motion at a high level of abstraction by the use of a structured programming language.

RESUME

Cet article a pour but de décrire une méthode de représentation et d'animation de modèles articulés en trois dimensions. L'article propose d'une part, la définition d'un modèle de segments et d'articulations et d'autre part, la précision du mouvement de celui-ci à un niveau élevé d'abstraction en utilisant un langage structuré de programmation.

KEYWORDS: Computer animation, figure modelling, movement representation

1.0 INTRODUCTION

Many advances have been made in computer animation in the last few years, especially in the area of figure modelling and motion specification. Several methods have been proposed including the modelling and control of figures using procedures (procedural modelling) [7], the control of a physical model by the application of forces (dynamic modelling) [1], the use of goal-directed systems for the generation of a model's motion [5,10], and the use of key frame animation [3], one of the oldest animation techniques still in use.

A different approach for the modelling of a three-dimensional articulate figure and the subsequent control of its motions will be presented here. It permits a user to define a model (representing a real three-dimensional figure) consisting of segments and joints [11], and to specify the desired motion of its joints using a high-level structured programming language. The problem of figure modelling and motion specification is dealt with in terms of kinematics: the study of position (displacement)

and its time derivatives (velocity and acceleration). Considerations of force and mass (dynamics) [4,9], balance [8], and obstacle avoidance [6] are beyond the scope of this discussion.

2.0 DESCRIPTION OF MODELS

Before an attempt is made to specify a desired motion for a model, a method for specifying the model must be available. The model's individual rigid links (segments) are unspecified in this study. It is assumed, however, that the model's links can be defined as graphical objects, using a high-level graphics language, before the model is constructed.

2.1 DESCRIPTION OF JOINTS

A joint has up to three degrees of freedom, that is, it can be rotated about each of the X, Y, and Z axes. Joints may be restricted to one or two degrees of freedom by permitting the joint to rotate about only one or two of the axes. Thus, simple joints such as fingers (hinge joints), and complex joints, such as shoulders (ball-and-socket joints) can be simulated. A joint connects only two links. A joint can move independently of all other joints, hence the position of one joint does not affect the motion of another. The links are restricted in their movements about a joint. During a single joint's movement, one link (the primary link), is considered stationary and the second link (the secondary link) moves with respect to the stationary link. A single link can function as both a primary link and a secondary link if it belongs to two or more different joints. One link, the model's main link, is singled out from the others. All movement ultimately refers to the main link. Only one link may be designated as the main link and it must be the primary link in all joints it belongs to.

Each instance of a joint is assigned a unique identifier to permit subsequent motion specifications. The user may place restrictions on the range of angles through which a link may travel and may specify where the two links are

to be joined. A typical statement creating an articulate joint is

```
JOINT joint_identifier,
      primary_link,    relative_location_1,
      secondary_link, relative_location_2,
      x_extremes, y_extremes, z_extremes
```

where 'joint_identifier' is the unique identifier for this instance of a joint; 'primary_link' and 'secondary_link' may be either previously defined graphical objects (which have been defined in independent coordinate systems) or submodels. 'primary_link' is the stationary link with which 'secondary_link' moves. 'relative_location_1' and 'relative_location_2' are vectors which contain the relative positions, to each of the graphical object's coordinate system, where the joint is to be attached. The extreme parameters are component vectors which store a pair of extreme angles beyond which the joint can not move. Extremes, as well as the joint's current angles, are given relative to the joint's predefined neutral position of (0°, 0°, 0°).

2.2 INTERNAL REPRESENTATION

The model of an articulate figure is described by a tree structure of nodes and arcs. Links are represented by nodes and the joints are represented by arcs. Each segment is defined on its own local coordinate system [2,5,10]. The nodes of each level move with respect to the nodes of the higher level and are considered stationary by the nodes below. The nodes at the leaves of the tree represent the outermost extremities of the model; the root node is considered the main link. Figure 1a is an example of a partial model of a human figure. Figure 1b contains the model's tree structure. If a root node is no longer required to act as the main link, a new node can be assigned to that role by the statement

```
FIGURE model_name MAIN major_link_in_body
```

It is possible, therefore, to animate a model with a different main link in different scenes.

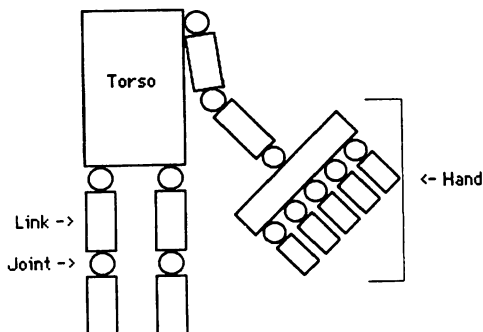


Figure 1a

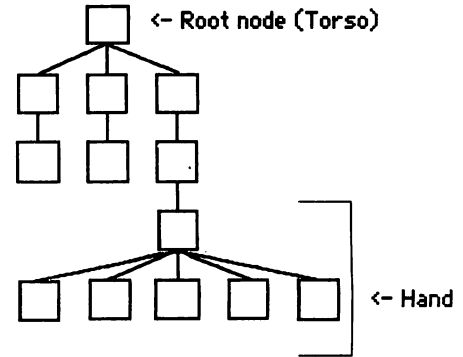


Figure 1b

- a) Partial model of a human figure
- b) Model's tree structure

Each node is associated with at least one arc, as in the case of the extremities which are secondary links. Typically, there are two arcs associated with each node, corresponding to a link (such as the femur of a leg) which acts as both a primary and secondary link. However, a node may have three or more arcs associated with it, as with the hand, which has six arcs (one representing the wrist joint, the other five representing the finger joints). A node may have at most n arcs associated with it. The branching factor n is theoretically unlimited, but a factor of 10 is deemed sufficient to define most articulate figures.

The tree structure permits the representation of open kinematic chains only. A kinematic chain is a linear sequence of links which are connected by joints. In an open chain, one end point is fixed and the remaining chain is allowed to move freely, as in Figure 2a. In a closed chain, more than one end point is fixed in space, as in Figure 2b. For example, if two hands are joined together and the arms are allowed to move while keeping the body motionless, a closed kinematic chain is formed by the arms. The motion produced in such a chain is more complex to analyze and is beyond the scope of this discussion.

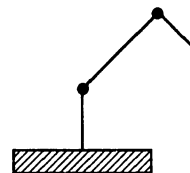


Figure 2a

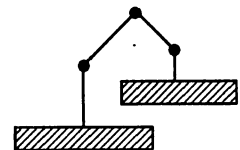


Figure 2b

- a) Open kinematic chain
- b) Closed kinematic chain

Each model has a table associated with it. The table contains information from the declaration of the model, in particular, the

identifier, the rotational extremes associated with each degree of freedom, the current rotational angles of each degree of freedom, and the instantaneous velocity and acceleration of the joints along each degree of freedom. The table describes the model completely at all times.

2.3 MODEL DEFINITION USING SUBMODELS

An articulate model can be defined using two basic techniques. The first allows the use of articulate submodels. Instead of simply using static graphical objects for each link, the secondary link can consist of a grouping of other links and joints, which permits the creation of intermediate models or submodels that can be referenced frequently. The statement

```
arm := JOINT 2, humerus, (1.0, 5.0, 1.0),
      forearm, (0.0, 0.5, 1.0),
      (0°, 135°), (0°, 180°)
```

creates a model of an arm consisting of a forearm and a backarm (humerus) joined at the elbow. The secondary link (forearm) is a model itself, consisting of links and joints. The relative location vector for the secondary link is given with respect to the main link in the model 'forearm'.

An advantage of this method is that symmetric models can be created with fewer statements. For example, creating a model of a human body first entails the creation of submodels for the right arm and the right leg. Once defined, each submodel can be duplicated and joined to both the right and left half of a human torso. A separate set of left limbs need not be defined. A problem arises here, however. The values of the joint identifiers should not be duplicated for the right hand and left hand limbs, or else the two sets of limbs will behave identically. Thus, the submodel joint identifiers must be modified before the submodels can be connected to the torso, to ensure unique identifiers. Consider the statement

```
arm := JOINT 2, humerus, (1.0, 5.0, 1.0),
      forearm <TRANS BY 10>,
      (0.0, 0.5, 1.0),
      (0°, 135°), (0°, 180°)
```

which creates a model of an arm consisting of a forearm and backarm (humerus) joined at the elbow. The secondary link (forearm), which was previously defined, is an articulate model whose joint identifiers have been mapped onto a new range of identifiers starting with the identifier 10.

2.4 MODEL DEFINITION BY CONSTRUCT

The submodel approach may produce temporary models unnecessarily. For this reason, an alternate technique for creating models is provided which assumes that none of the model's subcomponents is required subsequently. This

method creates only the resultant model with no intermediate articulate models. The following construct is an example

```
MODEL arm
JOINT 1, palm, (0.0, -3.0, 1.0),
      little, (0.0, 1.5, 0.5),
      (-90°, 45°), (-45°, 45°)
JOINT 2, palm, (0.0, -3.0, 2.0),
      ring, (0.0, 1.5, 0.5),
      (-90°, 45°), (-45°, 45°)
JOINT 3, palm, (0.0, -3.0, 3.0),
      middle, (0.0, 1.5, 0.5),
      (-90°, 45°), (-45°, 45°)
JOINT 4, palm, (0.0, -3.0, 4.0),
      index, (0.0, 1.5, 0.5),
      (-90°, 45°), (-45°, 45°)
JOINT 5, palm, (0.0, -3.0, 5.0),
      thumb, (0.0, 1.5, 0.5),
      (-90°, 45°), (-45°, 45°)
JOINT 6, ulna, (0.0, -5.0, 1.0),
      palm, (0.0, 3.0, 1.0),
      (-90°, 0°), (-25°, 25°)
JOINT 7, humerus, (0.0, 5.0, 1.0),
      ulna, (0.0, 0.5, 1.0),
      (0°, 35°), (0°, 180°)

ENDMODEL
```

If the model 'hand' comprising the first five joints has been defined prior to the use of the MODEL construct, then the model definition can be simplified:

```
MODEL arm
JOINT 6, ulna, (0.0, -5.0, 1.0),
      hand, (0.0, 3.0, 1.0),
      (-90°, 0°), (-25°, 25°)
JOINT 7, humerus, (0.0, 5.0, 1.0),
      ulna, (0.0, 0.5, 1.0),
      (0°, 35°), (0°, 180°)

ENDMODEL
```

3.0 DESCRIPTION OF MOTION

A model can be animated by a number of different approaches. Conventional animation relies heavily on two-dimensional techniques such as key frames and interpolation ("in-betweening"), while computer animation usually expresses position and velocity as functions of time. Key frames are the frames used to provide the information which express the proper effects of movement. In animation studios, key frames are drawn by the head animators, while the frames required to create the smooth animation are produced by the in-betweeners.

One of the earlier approaches to computer animation consisted of having the computer assume the role of the in-betweeners [3]. While some very effective animations have been achieved, in-betweening is two-dimensional in origin and awkward to apply to three-dimensional figures. In-between frames are frequently linearly interpolated, resulting in temporal discontinuities and movements which only approximate actual trajectories, thus, deforming the animation.

Functions of time are evaluated on a frame-by-frame basis which involves specifying a path over time. This method has the advantage of producing motion with few temporal discontinuities. The description of a three-dimensional path as a function of time, however, is generally a difficult task.

3.1 MOTION SPECIFICATION

The proposed approach for motion specification treats motion somewhat differently than either the key frame or functional technique. Whereas key frame animation views a figure with an external perspective and the functional approach views a figure from a piecewise perspective, the proposed technique treats the model as a unit and views it from an internal perspective (i.e. from the model's point of view). Therefore, a model's positional orientation can be specified throughout time. Similarity exists with key frame animation since key frames or positional extremes are used throughout time, however, each of the model's joints is employed and dealt with on a high level of abstraction, thus allowing more flexibility in the model's animation.

The motion for a single joint is specified by the joint identifier, an initial starting position (angle), and a set of key frames (movements). Each movement contains the frame identifier at which the movement is completed, the position (angle) of the joint at the frame, and the interpolation method used to reach this positional extreme.

The position angles relate to a neutral position, arbitrarily defined as (0°, 0°, 0°). The neutral position is defined with respect to the primary link in the joint. The frame identifiers represent the number of ticks (units of time) which have passed during the motion sequence. The frame identifiers are given with respect to the start of the motion specification. The initial frame is arbitrarily assigned the value of zero. The interpolation techniques available are: linear, acceleration, deceleration, and a combination of both acceleration and deceleration.

The parameters above are sufficient to specify a single joint's movement throughout an animation sequence. Temporal discontinuity problems can arise if the interpolation method is not chosen carefully. The system can avoid these problems by determining a joint's instantaneous velocity and acceleration before the interpolation, thus making adjustments to the selected interpolation. This results in smaller temporal discontinuities.

A complete motion definition for a model consists of motion specifications for all of the joints present in the model. The specifications are independent of each other, therefore, there may be different numbers of key frames for each joint. All the joint key frames must, however, end at the same unit of time.

3.2 TREE TRAVERSAL

Before a model's tree structure is traversed, the symbol table is updated with each joint's current positional angle, and its current velocity and acceleration. This information is obtained from both the motion specification and the interpolation routines. The tree structure is completely traversed each time the model is displayed (i.e. once each frame). The traversal algorithm is a simple recursive post-order routine. It assembles each model's instance (from the extremities inwards) with respect to the main link in the model. The resulting instance is displayed by the high-level graphical language. Recursive routines are employed in the tree traversal because they allow storage of the the model's primary-secondary link relationship in the recursion stack. Also, they allow the use of the same routines on models whose main link (root) has been changed.

3.3 EXPLICIT DEFINITION

A motion can be specified using different approaches. There are two different methods for explicitly defining a motion. The first technique involves the use of a construct for the definition of a motion. A typical example is

```
MOTION motion_name
  JOINT joint_id
    POSITION angle_x, angle_y, angle_z
    FRAME frame_id
      POSITION angle_x, angle_y, angle_z
      INTERPOLATE interpolation_technique

    FRAME frame_id
      POSITION angle_x, angle_y, angle_z
      INTERPOLATE interpolation_technique
    . . .
  ENDJOINT

  JOINT joint_id .....
    FRAME ...
    . . .
  ENDJOINT
  . . .
ENDMOTION
```

With the construct, the exact definition of the motion can be specified. 'motion_name' names the set of joint-motion specifications. The joint identifiers correspond to those defined in the model. Each joint can have at most n key frames, where n is a predetermined value. The construct permits a structured approach to producing a motion specification. It, however, does not provide for the use of other control constructs.

A less structured method is also available. A typical example of the definition technique is

```
motion_name [joint_id, key_frame_id] :=
  FRAME frame_id
    POSITION angle_x, angle_y, angle_z
    INTERPOLATE interpolation_technique
```

The approach allows direct access to the motion variable. It can be employed within other constructs and it permits the use of iteration to produce the motion specifications, thus reducing the number of statements needed and the amount of work needed to define the specifications.

3.4 IMPLICIT DEFINITION

Once a motion has been defined, the specification can be viewed as a unit (motion primitive), thus it can function as a building block for the definition of more complex motions. A primitive motion algebra has been introduced for this use. For example, to create a motion which enables a human model to hop, skip, and jump 10 times, the following statement can be used:

```
new_walk := 10 * (hop + skip + jump)
```

'new_walk' is the resulting action. 'hop', 'skip', and 'jump' are previously defined motions which allow a model to hop, skip and jump respectively. The constant 10 is the repetition factor that is applied to the actions 'hop', 'skip', and 'jump'.

This technique relies heavily on the availability of predefined motion primitives. It is recognized that the explicit definition of such primitives can be both difficult and time consuming, therefore, operations have been introduced which allow a more convenient definition of the motion primitives. For example, if a motion primitive (walk) exists which allows a model to walk, it may be desirable to employ the action of the legs in a different motion. The operation STRIP has been introduced, which creates a partial motion primitive from a given motion. In the statement

```
walking_legs := STRIP walk, 5, 6, 7, 8, ...
```

'walking_legs' is a new motion primitive which when applied to a model, animates the legs. The values 5, 6, 7, 8, ..., are the joint identifiers present in the model's legs. Using this method, several motion primitives can be created that animate only portions of a given model.

The operation SYNCHRONIZE has been introduced, which when given a set of partial motions, creates a new motion that animates portions of the model concurrently. The statement

```
my_walk := SYNCHRONIZE walking_legs,  
                    swinging_arms
```

defines a motion primitive (my_walk) which causes the figure to walk while swinging its arms. This approach allows an increase in the number of motion primitives, but at a much higher level of abstraction.

Such an approach to the creation of motion gives rise to a large number of operations which can tailor motion primitives to the needs of the

current animation sequence. Possible operations include a scaling factor which stretches or shrinks a motion primitive temporally, a scaling factor which modifies the amplitude of a motion's movement, and a negation which reverses the order of the movements present in a motion primitive. These operations, as well as the operations * and +, have the advantage that intimate knowledge of the model's structure and motion definitions are not necessary. They allow the use of motion at a high level of abstraction.

3.5 USE OF A MOTION DEFINITION

Once a model is created and motions have been specified, a model can be animated.

```
ANIMATE model_name FROM start_frame TO end_frame  
        USING motion_name
```

The frame identifiers are given relative to the start of a scene. If the motion does not span the entire animation sequence, the motion will automatically repeat until the time span is covered.

Several models of a class (i.e. models of identical structure) may be animated by the same motion. Models of the same class and with different-sized links will react identically. Models with an equal number but with different types of joints can still be driven by these motion specifications, however, the resulting model movements may be difficult to predict. The models place restrictions on their movements. For example, if a rotation is employed which violates a joint's extremes, the joint will enforce its extreme rotation limits over the motion specification given. This allows two models of the same class, but different restrictions, to behave realistically using the same motion set.

Once all elements for a scene have been defined, it can be explicitly specified with static and dynamic components. A typical statement creating a scene is:

```
SCENE scene_identifier  
    DISPLAY background, ...  
    DISPLAY any static models  
  
    . . .  
  
    ANIMATE model_name FROM ....  
    ANIMATE any dynamic models  
  
    . . .  
  
    movements which vary with time  
    (camera movements, panning,  
    zooming, etc.)  
  
    . . .  
  
ENDSCENE
```

The resulting scenes are displayed by the statement:

SHOOT SCENES

The scenes are displayed in sequential order ranging from the lowest scene identifier to the highest one. The use of scene constructs is advantageous because the animation sequence can be broken into a set of independent scenes, thus allowing its construction on a piecewise basis. The creation of independent scenes also permits changes in the scene ordering without any knowledge of the frame identifiers involved.

4.0 ENHANCEMENTS

The system is designed as an extension to a host language, to be executed in a batch environment. A preprocessor translates all extended language statements into procedure calls. Other implementations, e.g. as a command language to be interpreted by an executive kernel, are possible. The batch environment creates problems in the use of both predefined models and motions. At present, these definitions must be made at the beginning of a program before they can be employed. Ideally, there should be a method to save any models and motions defined in a program. Therefore, subsequent programs would need only load the definitions for models and motions before using them. This would allow the construction of a library of models and their movements.

A problem with the use of library definitions is that unless a user has defined a model and its motions, or has previously worked with them, it is difficult to determine how they will appear. It would be useful to have a viewing utility which permits the viewing of predefined models and motions in an interactive environment. This would allow the user to determine exactly what had been previously defined and what further definitions must be made in the program.

5.0 CONCLUSION

The project has several advantages over existing systems. It permits the modelling and animation of figures at a high level of abstractions. Where many systems have problems working with rotations, this system effectively deals with rotations. The use of rotations enables the creation of generalized motions which can be applied to more than one model, while motion specifications using paths and forces can only be used on the specific models for which they were designed. Interpolation currently works with articulate figures at the point level since it only has access to the two-dimensional projection of a figure. The approach presented deals with interpolation on a rotational basis (i.e. the rotational extremes correspond to the key frames in two-dimensional interpolation), thus allowing the animation of three-dimensional figures.

6.0 REFERENCES

- [1] Armstrong, W.W. and M. Green, The Dynamics of Articulated Rigid Bodies for Purposes of Animation, Graphics Interface '85, 1985, pp. 407-415.
- [2] Badler, N.I. and S.W. Smoliar, Digital Representations of Human Movement, ACM Computing Surveys, Vol. 11, No. 1, 1979, pp. 19-38.
- [3] Burtnyk, N. and M. Wein, Computer-Generated Key-Frame Animation, Society of Motion Picture + TV Engineers, Vol. 80, 1971, pp. 149-153.
- [4] Horn, B.K.P., Kinematics, Statics, and Dynamics of Two-Dimensional Manipulators, Artificial Intelligence: An MIT Perspective, Vol. 2, 1979.
- [5] Korein, J.U. and N.I. Badler, Techniques for Generating the Goal Directed Motion of Articulated Structures, IEEE Computer Graphics and Applications, Vol. 2, No. 9, 1982, pp. 71-81.
- [6] Perez, L. and M.A. Wesley, An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles, Communications of the ACM, Vol. 22, 1979, pp. 560-570.
- [7] Magnenat-Thalmann, N. and D. Thalmann, Actor and Camera Data Types in Computer Animation, Graphics Interface '83, 1983, pp. 203-207.
- [8] McGhee, R.M., Control of Legged Locomotion Systems, Proc 1977 Joint Automatic Control Conference, Vol. 1, pp. 205-213.
- [9] Paul, R.P., Robot Manipulators, MIT Press, Cambridge, Mass., 1981.
- [10] Zeltzer, D., Motor Control Techniques for Figure Animation, IEEE Computer Graphics and Applications, Vol. 2, No. 9, 1982, pp. 53-59.
- [11] Zeltzer, D., Representation of Complex Animation Figures, Graphics Interface '82, 1982, pp. 205-211.

CONSTRAINT-BASED MODELING OF THREE-DIMENSIONAL SHAPES

Przemyslaw Prusinkiewicz and Dale Streibel

Department of Computer Science
University of Regina
Regina, Saskatchewan, Canada S4S 0A2

ABSTRACT

This paper shows that constraint-based modeling, so far perceived primarily as a graphics technique for man-machine interaction, also provides a viable method for the modeling of complex surfaces. The idea of constraint-based modeling of three-dimensional shapes is described and illustrated by examples. Difficulties related to the practical application of this idea are discussed, and methods for overcoming them are outlined. A potential of the constraint-based approach to the modeling of shapes found in nature is indicated.

RESUME

Dans les systèmes infographiques à contraintes développés jusqu'à présent, les contraintes géométriques étaient utilisées surtout en qualité d'une technique d'interaction. Cependant, les mêmes contraintes peuvent former aussi une base pour modéliser des objets à trois dimensions. Cet article présente l'idée principale du modelage à l'aide des contraintes et l'illustre avec des exemples. L'application de la méthode pour modéliser des surfaces complexes est mise en évidence. Les problèmes numériques associés sont discutés, et une technique pour les atténuer par une décomposition hiérarchique du modèle est introduite. L'application potentielle de la méthode pour modéliser des formes de nature est indiquée.

Keywords: constraint-based modeling, polygon meshes, free-form surfaces.

1. INTRODUCTION

Of all the constraints of Nature, the most far-reaching are imposed by space.

Peter Stevens, *Patterns in nature*.

One common computer graphics technique for representing three-dimensional objects uses polygon meshes. A mesh is defined as a set of connected, polygonally bounded planar surfaces. Polyhedra are examples of meshes, but the notion of a mesh is more general. In particular, it also includes polygonal approximations of curved surfaces.

A mesh description consists of a specification of vertices, edges and faces. Known methods of mesh description require the positions of all vertices to be explicitly specified in a system of coordinates [Foley and van Dam 1983]. This is convenient in many situations, for instance, when a mesh is rendered. However, other parameters may be more convenient to use when a mesh is modeled. For example, consider descriptions of a regular tetrahedron. In terms of edges its definition is trivial - the tetrahedron must have four edges of equal length. In contrast, the description of a regular tetrahedron in terms of vertices is by far less intuitive, since their coordinates cannot be specified without arduous calculations.

Mesh definition by specifying the lengths of edges falls into the category of constraint-based modeling. Instead of specifying vertices directly, a set of constraints, or relations between vertices, is defined. The idea of specifying geometric figures using constraints is not new to computer graphics. It was first implemented in Sketchpad [Sutherland 1963], and followed in several other interactive graphics systems [Knuth 1979, Borning 1981, Van Wyk 1982, Nelson 1985]. Due to its intuitive character, constraint-based modeling was used there primarily as the basis for man-machine interaction. The possibility of building a constraint-based system for the purpose of computer aided design was indicated by Lin, Gossard and Light [1981].

This paper presents a new application of constraint-based modeling - definition of complex three-dimensional shapes.

2. UNSTRUCTURED MODELING

Various types of constraints can be used when describing a mesh. For example, they may characterize vertices as co-linear or co-planar, specify areas of faces, fix the angles between edges and faces, etc. The mesh representation described in this paper uses distances between points as the main form of constraint. Additionally, lines can be specified as parallel to any plane of the system of coordinates (xy , xz or yz), and selected coordinates of vertices can be explicitly given. Explicit specification of some coordinates and directions is necessary to position a rigid object in space, so that it cannot translate nor rotate. Thus, the complete description of a mesh containing n vertices consists of:

Table 1. A constraint-based definition of a regular tetrahedron.

Specification	Comment
$x_A = y_A = z_A = 0$	Vertex A lies in the origin of the system of coordinates.
$y_B = z_B = 0$	Vertex B lies on the axis x.
$z_C = 0$	Vertex C lies on the plane xy.
$d(A,B) = d(B,C) = d(C,A) =$ $d(A,D) = d(B,D) = d(C,D) = L$	Distance between any two vertices is equal to a given constant L.
$e1: A-B$ $e2: B-C$ $e3: C-A$ $e4: A-D$ $e5: B-D$ $e6: C-D$	List of edges.
$p1: e1-e3-e2$ $p2: e1-e5-e4$ $p3: e2-e6-e5$ $p4: e3-e4-e6$	List of polygons.

- A system of $3n$ equations with $3n$ unknown vertex coordinates. Each equation represents a constraint.
- A list of edges expressed in terms of vertices.
- A list of polygons expressed in terms of edges.

An example of a constraint-based definition is given in Table 1.

In order to find the coordinates of the vertices, the system of constraints must be solved. Since the equations describing distances are quadratic, only numerical methods can provide a general solution. The results presented in this paper were obtained using the Newton method [Conte 1965]. For example, Fig. 1 shows a tetrahedron resulting from the description given in Tab. 1. Another example - a cubo-octahedron - is shown in Fig. 2.

The approach to constraint-based modeling described above is called unstructured, because all constraints are combined into one large system of equations and solved simultaneously. In practice, this approach presents several difficulties. The first difficulty occurs when defining a mesh. If the constraints are not correctly chosen, the resulting mesh will not be rigid, or will contain dependent (redundant) constraints. In both cases, the Newton method will fail to provide a solution (the Jacobian is equal to zero). Proper selection of constraints is a nontrivial task, because the rigidity of an object may depend on particular values of the edge

lengths. A simple example, taken from [Hain 1967], illustrates this in the two-dimensional case (Fig. 3).

Even if the set of constraints is correct, its solution may be difficult to find: for a given set of initial values, the numerical method need not converge or it may converge to a wrong solution. This second situation occurs, if more than one object satisfies the given constraints. Unfortunately, this is often the case. For example, even the simple description of a tetrahedron which has been presented in Tab. 1 allows for 8 different solutions: the base ABC can be placed in any of the four quadrants of the plane xy , and the vertex D can be located either above or below this plane.

The following section describes a technique for overcoming these difficulties.

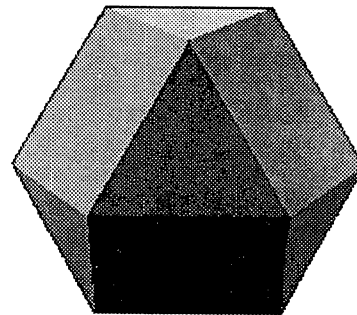


Fig. 2. A cubo-octahedron.

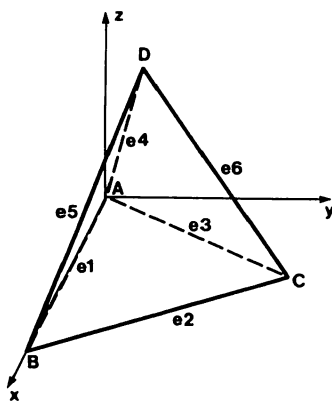


Fig. 1. A tetrahedron described by Tab. 1.

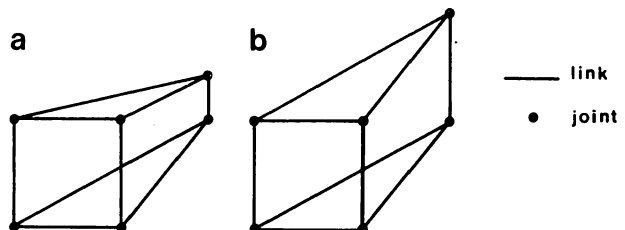


Fig. 3. Rigidity of an object may depend on particular values of distances. Planar meshes (a) and (b) differ only by the lengths of some edges; however, mesh (a) is rigid, while mesh (b) is not.

3. STRUCTURED CONSTRAINT-BASED MODELING

The correct solution can be more easily found if the mesh being defined can be thought of as the last element in a sequence of rigid submeshes. The first element in this sequence is a rigid object, called the kernel, simple enough to be properly defined and solved. The subsequent submeshes differ from each other by a few additional vertices and edges. Thus, the submesh S_{i+1} contains S_i as its proper subset. When calculating S_{i+1} , all vertices of S_i are already known, so that at each stage only a small system of equations has to be solved. Consequently, the sequence of submeshes S_i imposes a structure on the set of constraints.

As an example of the above idea, consider the construction of a pyramid from horizontal slabs. The definition of the slab is given in Fig. 4. After fixing the coordinates of the base, the construction progresses by placing consecutive slabs on top of each other, until the top point is formed (Fig. 5).

In the case of the pyramid, the length of the horizontal edges decreases from one slab to the next one by a constant value: $a_{i+1} = a_i - c$, $c > 0$. The length of the slanted edges b_i is constant. By expressing the lengths of edges using other formulas, the pyramid can be deformed, and more complex shapes can be obtained. For example, Fig. 6 shows an Eiffel-Tower-like shape resulting from decreasing the length of the horizontal edges of the mesh by a constant factor: $a_{i+1} = a_i \cdot c$, $0 < c < 1$. Fig. 7 shows a dome-like shape obtained using septagonal slabs. In this case, the lengths of both the horizontal and the slanted edges are changed according to the formulas:

$$a_{i+1} = \sqrt{a_i^2 - c^2 i^2} \quad b_{i+1} = \alpha \sqrt{(a_i - a_{i+1})^2 + c^2}$$

Values of the constants α and c are chosen in such a way that the dome can be inscribed in a sphere. Finally, Fig. 8 shows a vase obtained by changing the length of the horizontal edges according to the function $a_{i+1} = a_i + c \cdot \cos(i\omega)$, with $c, \omega > 0$. The length of the slanted edges is constant, and the slabs are septagonal.

Unfortunately, not every mesh can be decomposed into a sequence of rigid submeshes. Fig. 9 illustrates this in the two-dimensional case: the whole mesh is rigid, but it does not include any rigid submesh. Consequently, no kernel (other than the entire mesh) can be distinguished. Nevertheless, in many practical cases the decomposition is not only

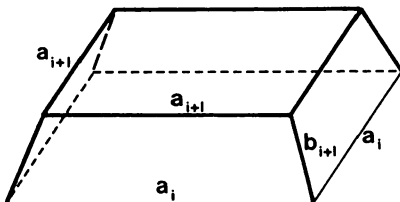


Fig. 4. Definition of a slab.

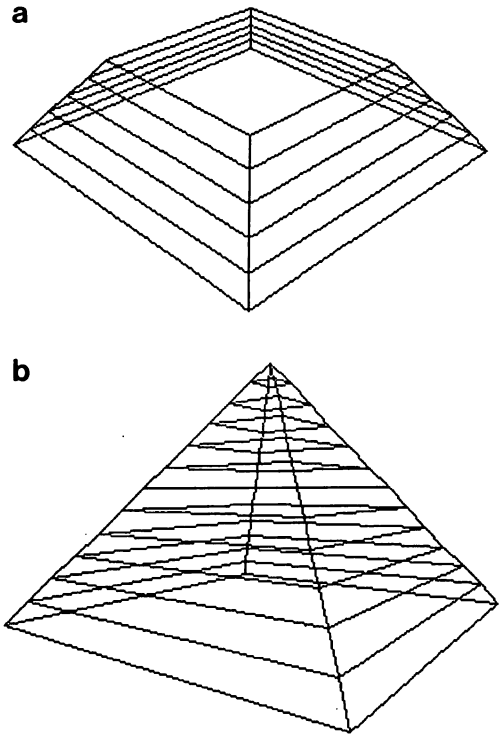


Fig. 5. Construction of a pyramid from slabs. (a) Construction in progress. (b) The final pyramid.

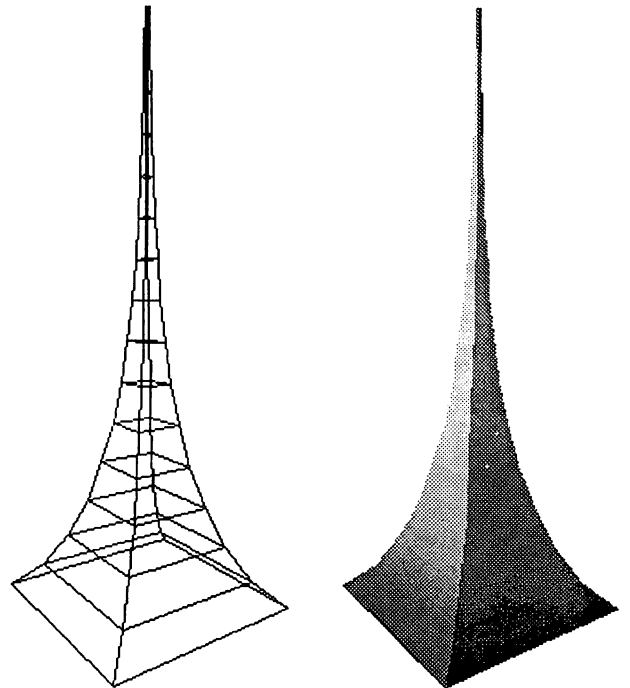


Fig. 6. The "Eiffel tower".

feasible, but it results in a straightforward way from the mesh description. This happens, when graphic modeling can be thought of as a process similar to building a real-life construction. Usually, consecutive phases of a construction correspond to rigid objects, because a construction becoming rigid only in a late phase of development would be technologically difficult to make. Presumably, this argument applies not only to man-made constructions, for example found in architecture, but also to natural objects such as crystals or living organisms, which result from a growth process.

Just as the varying length of horizontal segments determines the shape of the vase, a varying growth rate may determine a shape created by Nature. This point is best described by Stevens [1974]:

No matter how we try, we cannot make a saddle from five equilateral triangles or a simple cup from seven... . Nature too is similarly constrained. She makes cups and saddles not as she pleases but as she must, as the distribution of material dictates... . If the perimeter of a shell grows at a faster rate than the center, the perimeter curls and wrinkles. No genes carry an image of how to place the wrinkles; no genes remember the shape of the shell; they only permit or encourage faster growth at the perimeter than at the center.

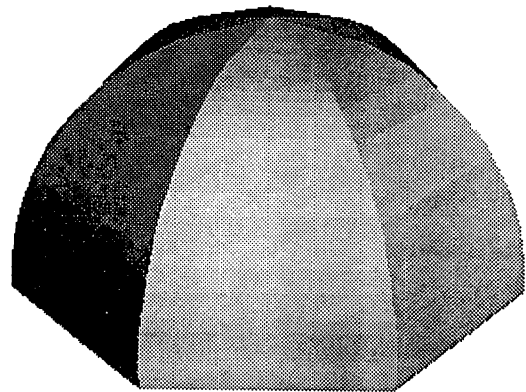
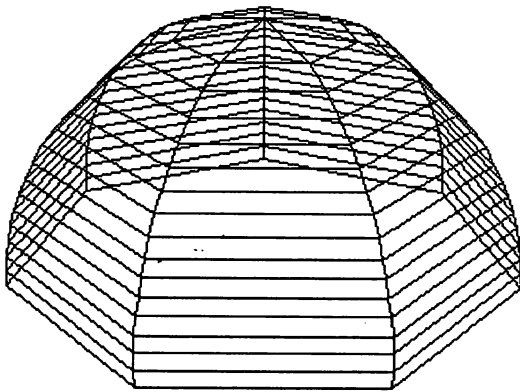


Fig. 7. A dome.

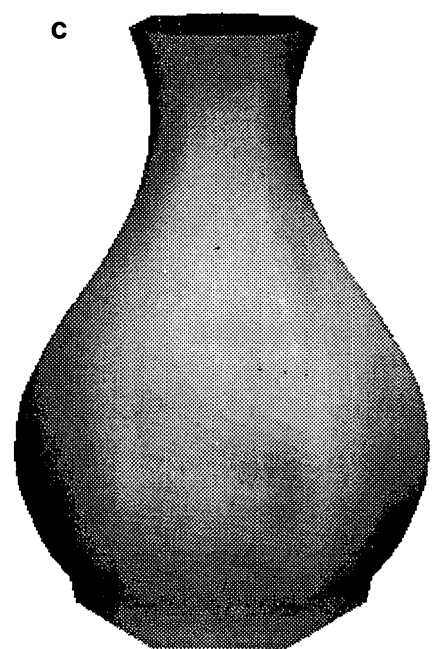
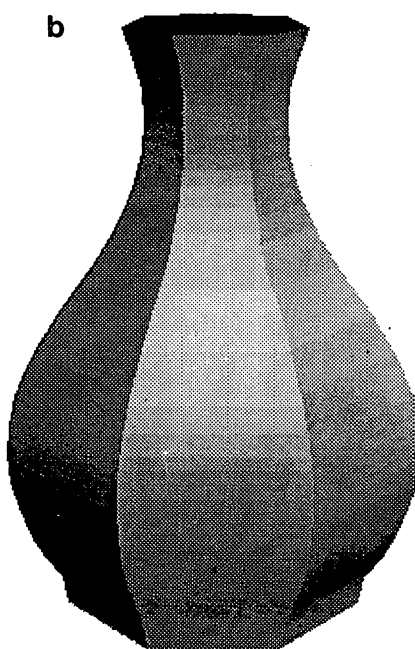
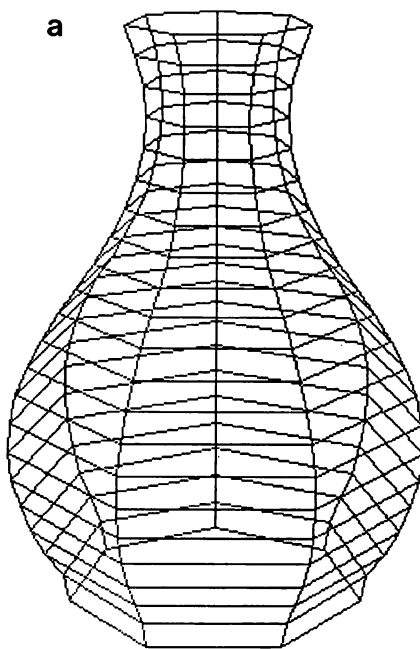


Fig. 8. A vase. Figures (b) and (c) represent two different renderings of the polygon mesh (a).

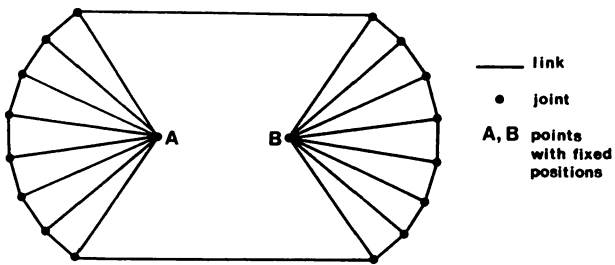


Fig. 9. Example of a planar mesh with no rigid submesh.

Since structured constraint-based graphics modeling may imitate the natural process of growth, it appears to have great potential as a method for modeling shapes found in nature. An example of a shell modeled using the constraint-based approach is shown in Fig. 10. First, a planar section of the shell is "grown" by adding consecutive trapezoidal compartments (a). This provides a basis for creating three-dimensional "top" (b) and "bottom" portions of the shell. The two parts are connected together to form the complete polygon mesh (c). The final shape is shown in Fig. (d).

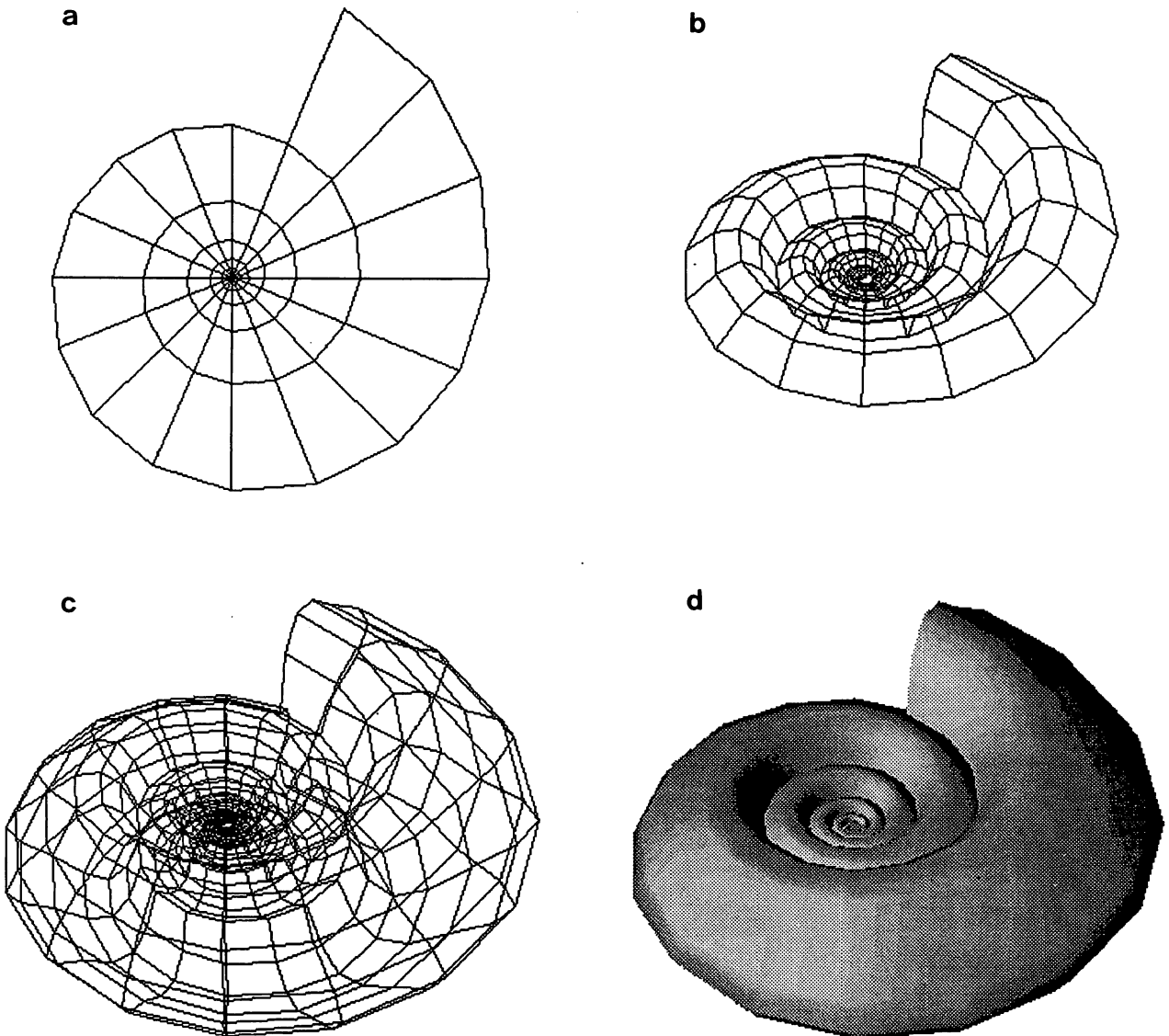


Fig. 10. Construction of a shell.

4. CONCLUSIONS

This paper presents a new method for defining three-dimensional shapes. It relies on constraint-based modeling of polygon meshes. The definition of an object in terms of constraints can be more straightforward and simpler than other types of definitions. In order to find vertices of a mesh defined using constraints, a system of nonlinear equations must be solved. In the general case, only numerical methods can be used for this purpose. Solving the system of equations can be made easier, if the mesh can be decomposed into rigid submeshes. Such decomposition is possible in many practical situations. Application of the constraint-based approach to the modeling of natural objects, for example flowers and shells, is an attractive topic open for further research.

ACKNOWLEDGMENT

This research was partially supported by grant number A0324 from the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

- Borning, A. [1981]: The programming aspects of Thinglab, a constraint-oriented simulation laboratory. *ACM Trans. on Programming Languages* 3, No. 4, pp. 353-387.
- Conte, S. D. [1965]: *Elementary numerical analysis: An algorithmic approach*. McGraw-Hill, New York.
- Foley, J. D., and van Dam, A. [1983]: *Fundamentals of interactive computer graphics*. Addison-Wesley, Reading.
- Hain, K. [1967]: *Applied kinematics*. McGraw-Hill, New York.
- Knuth, D. E. [1979]: *TEX and METAFONT*. Digital Press and American Mathematical Society, Bedford.
- Lin, V. C., Gossard, D. C., and Light, R. A. [1981]: Variational geometry in computer-aided design. *Computer Graphics* 13, No. 3, pp. 171-177.
- Nelson, G. [1985]: Juno, a constraint-based graphics system. *Computer Graphics* 19, No. 3, pp. 235-243.
- Stevens, P. S. [1974]: *Patterns in nature*. Little, Brown and Co., Boston.
- Sutherland, I. E. [1963]: *Sketchpad, a man-machine graphical communication system*. In *1963 Spring Joint Computer Conference*, reprinted in Freeman H. (Ed.): *Interactive Computer Graphics*, IEEE Computer Soc. 1980, pp. 1-19.
- Van Wyk, C. J. [1982]: A high-level language for specifying pictures. *ACM Transactions on Graphics* 1, Nr. 2, pp. 163-182.

The Stochastic Modelling of Trees

Alain Fournier
David A. Grindal

Computer Systems Research Institute
Department of Computer Science
University of Toronto
Toronto, Ontario
M5S 1A4

ABSTRACT

We present here a fast method for the modelling of trees which brings together two interesting techniques. The trees are modelled as convex polyhedra for the description of the gross shape, and three-dimensional texture mapping is used for the detailed features.

The "essential" volume of the tree is represented as the convex intersection of half spaces. The advantage of this representation is that it allows an adaptive level of detail in the display. We use a special algorithm for the display of the convex intersection which computes it directly in the frame buffer. The algorithm also allows the computation of intersecting polyhedra.

To transform the convex polyhedra into a more realistic representation of trees, we use three-dimensional texture mapping to "modulate" the shape and the colour of the basic polyhedra. We then obtain an irregular non convex object, which is consistent in shape and general appearance regardless of the point of view and the size on screen. Three dimensional fractional Brownian motion is one of the procedural texture used.

KEYWORDS: tree modelling, half space intersection, stochastic modelling, frame buffer algorithms, adaptive modelling.

RESUME

Nous présentons ici une méthode rapide pour le modelage des arbres qui réunit deux techniques intéressantes. Les arbres sont modélés par des polyèdres convexes pour la représentation de la forme globale, et une texture à trois dimension est utilisée pour modeler les détails.

La forme "essentielle" de l'arbre est réalisée par des polyèdres convexes, résultats du calcul de l'intersection de demi-espaces. L'avantage de cette représentation est qu'elle permet un niveau adaptif de détail. Nous avons développé un algorithme pour le calcul de l'intersection convexe directement dans la

mémoire d'image. Avec une légère modification l'algorithme permet le calcul de polyèdres qui s'intersectent.

Nous transformons les polyèdres en une représentation plus réaliste des arbres en utilisant le "mapping" d'une texture à trois dimension pour moduler la forme et la couleur des polyèdres de base. Nous obtenons ainsi un objet non-convexe et irrégulier, dont la forme et l'apparence générale est consistante indépendamment du point de vue et de la taille de l'arbre sur l'écran. Le mouvement Brownien fractionnel à trois dimensions est une des procédures de génération de texture utilisées.

MOTS CLES: modelage d'arbres, modelage stochastique, intersection de demi-espaces, algorithmes de mémoire d'image, modelage adaptif.

1. Motivations

Trees are obviously very important in the modelling of natural scenes and landscapes. Problems are caused by the large number of trees needed and their considerable variety of shapes. The main criteria for a good model are to be realistic, easy to compute (both in terms of the basic operations needed and of the time complexity), flexible (capable of generating the intra- and inter-species variations in shape), adaptive (generating various level of details as needed) and compact. Of course, depending on the application, one or more of these criteria can be relaxed if not all can be met. The techniques used so far include grammar generation systems [AoKu84, Smit84], particle systems [ReB185], polygonal description plus two-dimensional texture mapping [Blo85] and simple volume primitives plus two-dimensional texture mapping [Gard84, Gard85]. The technique we will describe here, which is close in spirit to the ones used by Gardner, uses simple volume primitives (convex polyhedra) computed in the frame buffer associated with stochastic three-dimensional texture mapping. Table 1 gathers a subjective evaluation of these different techniques with regard to the above criteria. A scale of 0 (not at all) to 5 (best possible) is used.

Refs.	Real.	Easy (ops)	Easy (time)	Flex.	Adapt.	Compact
[AoKu84]	4	2	2	4.5	3	4
[ReBl85]	4.5	2	2	3.5	2	4
[Bloo85]	4.5	2	2.5	1	1	1
[Gard84]	3.5	2	3	2	3	4
Here	2	4	4.5	3	4	4

Table 1. Subjective comparison of tree models

To achieve flexibility, we will use a mixture of generative techniques, as in [AoKu84, Smit84] and stochastic techniques, as in [FoFC82, Reev83, ReBl85]. The goal of compactness will therefore be achieved, since the actual description for each tree is very small. We will have to pay special attention to the problems of *consistency*, that is keeping the appearance constant as the level of detail, the point of view and the size on screen of the displayed objects are changed.

It is highly desirable that the entire process of generating, rendering, and colouring the tree(s) be done in a reasonable time and with moderate amounts of computing power. Since our goal here is not *ultra-realism* but a balance between realism and time, we hope to be able to approach the conditions for real-time display. The system described here will not create images in real-time. However, it should be possible, with hardware and minor software improvements, to bring it close to or achieve real-time performance.

2. The three-dimensional shape

Primitives to model three-dimensional objects range from points to lines to polygons to higher degree surfaces. Most of these have been used to model trees. Polygons, because they are linear objects, and because most rendering systems ultimately deal with polygons at the display level, are a tempting choice. They have many drawbacks, however. Many polygons are needed to represent a complex shape, such as of a tree, and they constitute a very inflexible model, hard to parametrise or modify adaptively. There is another representation scheme which has most of the qualities of polygonal models and some additional advantages. The volume of the tree can be represented as the *convex intersection of half-spaces*. A *half-space* is the area of space all on one side of a plane. Formally, a half-space HS_i is the locus of points (x,y,z) such that $a_i x + b_i y + c_i z + d_i \geq 0$. If several half-spaces are intersected, the result is $V = \bigcap_i HS_i$ or

$$V = \left\{ (x,y,z) \in \mathbb{R}^3 : \forall i \ a_i x + b_i y + c_i z + d_i \geq 0 \right\}. \quad \text{This}$$

volume, usually enclosed, is convex, and its faces are all convex polygons.

This form of representation is rather different from any conventional means of storing three dimensional polyhedra. The most radical departure from the norm is that it does not store the vertices of the polyhedra. The only entities stored are the equations of the intersecting planes.

One benefit of storing planes is that there is added information stored in the equation. For the plane $ax + by + cz + d = 0$ the vector (a,b,c) is the normal to the plane. This fact will be used later, for the generation of the trees. It turns out that by using the normals, a user can create a wide range of trees easily and quickly. If the same normals and the same parameters are used, the procedure can also consistently generate the same tree.

Another advantage of the half space representation is its flexibility. When the object is defined by a set of half spaces, it is possible to get a finer representation by splitting the planes. This splitting can be done to any one plane, without greatly affecting the total volume or overall shape. With a polygon mesh it is a difficult process, because the criteria to merge and split polygons are not obvious, and a change can affect many polygon boundaries.

This scheme has another (minor) advantage over the polygon mesh, in that the amount of storage needed for the same polyhedra is a little less.

2.1. Generating the Tree

Using the normals (*ie* planes) to generate the trees gives more freedom in generating trees randomly. Many schemes could be thought of for splitting normals, in order to create a convex hull. In fact any grammar can drive the process. We will only describe one method here.

The principle is illustrated by Figure 1. The existing normal \vec{N}_1 defines the current plane P_1 . The normal \vec{N}_1 will be split into \vec{N}_2 and \vec{N}_3 , which will define the planes P_2 and P_3 . It is desirable that the area, or volume, described by P_2 and P_3 be approximately that described by P_1 . There should be some "natural" breakdown of the normals so that the end result after several splits, is roughly the same as the original plane.

In addition to the manner in which a given plane is split in two, there is the further choice of which plane is to be split. There are many possible rules which could be followed here. The "oldest" plane could be split each time. A "lifetime" could be assigned to each plane, with a probabilistic chance of it being split when its life is over (the most likely probability here would be a negative exponential), or planes could simply be chosen at random. In the system described here, the planes were split in generations. All the planes were split at each stage. Thus all the resulting planes are of the same "age", and there are always $2^n * N$ of them, where n is the number of generations and N is the number of initial planes. This is equivalent to applying the production rules of a parallel grammar at each generation.

The equation that governs the splitting of the normals can be read off of Figure 1. The normals should be split so that in the average case,

$$l_2 \cos \alpha_1 = l_3 \cos \alpha_2 = |BC|.$$

Since this equality only holds in the average case,

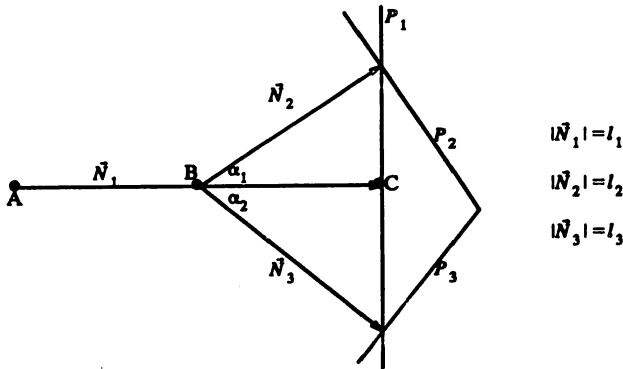


Figure 1 Splitting the Normal

there will be some random perturbation around the exact values. Even with this equation restricting the splitting method, there are still a great number of parameters to control. The following algorithm was used:

- Step 1) Choose point B. This point will be the origin for the two new normals, \vec{N}_2 and \vec{N}_3 . The parameters here are μ_B and σ_C . Point B will be chosen a distance down the normal from C: $|BC| = l_1 * \mu_B + \text{gauss}() * \sigma_B$.
- Step 2) Choose the angles at which the new normals will split from the present normal:

$$\alpha_1 = \mu_\alpha + \text{gauss}() * \sigma_\alpha$$

$$\alpha_2 = \mu_\alpha + \text{gauss}() * \sigma_\alpha$$

- Step 3) Choose the length of the two new normals:

$$l_2 = |BC| / \cos \alpha_1 + \text{gauss}() * \sigma_l * l_1$$

$$l_3 = |BC| / \cos \alpha_1 + \text{gauss}() * \sigma_l * l_1$$

The lengths are designed so that the end of the normal is in the plane P_1 .

- Step 4) Reduce the angle at which the new normals are created:

$$\mu_\alpha = \mu_\alpha * \text{ratio}$$

$$\sigma_\alpha = \sigma_\alpha * \text{ratio}$$

This maintains the user's control over the creation process. If the splitting angle were not reduced, then the normals resulting after two or three levels of recursion would have no resemblance to the original.

Since the representation being used is that of convex intersection, it is possible for one errant plane to chop the tree in half. This occurs if a normal is split far

enough away from its predecessor's original direction. The problem is roughly similar to that of self-intersection in two dimensional stochastic interpolation. Figure 2 demonstrates how this can happen if the normals split in just the wrong way. In fact, the problem occurs more often if the splitting is taking place in three dimensions (as is being done) instead of two dimensions (as is being shown). In the diagram the seven "outside" normals have been shortened for sake of clarity.

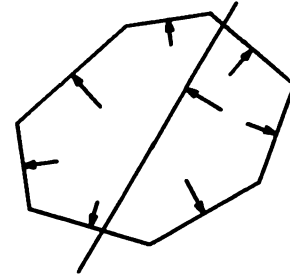


Figure 2 The Effect of One Errant Normal

In order to prevent this from occurring, one more restriction is added to the creation process. The procedure keeps only the normals that point "outwards". The algorithm ensures that if a normal's direction is into a certain octant, that the origin of the normal is also in that octant. If the normal is (a,b,c) and its point of origin is (x,y,z), then the normal is retained if and only if

$$ax \geq 0 \text{ AND } by \geq 0 \text{ AND } cz \geq 0.$$

This process of pruning the normals is demonstrated in two dimensions by Figure 3. In this diagram, normals \vec{a} , \vec{b} , and \vec{c} would be retained, where \vec{d} and \vec{e} would be rejected. Using this pruning method it can be seen that an occurrence such as that in Figure 2 is not possible. This means that the convex hulls should be fairly well proportioned. One "bad" normal can not cut away half of the volume.

The creation procedure lets the user define any number of normals to start. Empirically, it turns out that beginning with three to six normals gives the best results. This process gives a large amount of control over the result. If the input included a long vector, the result was usually a long thin tree. The input angles are additional parameters which permit wide control of the overall shape. In fact the sample space is large enough that it has not yet been fully explored.

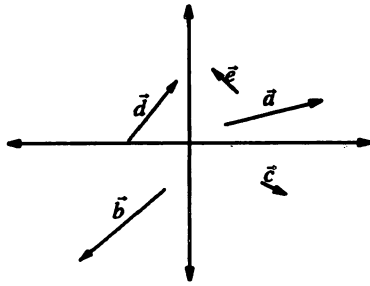


Figure 3 Example of Pruning the Normals

2.2. The Half-Space Intersection Algorithm

We now have a collection of planes to model the tree. What is needed is a visible-surface algorithm for the intersection of half-spaces. The problem of finding the convex intersection of half spaces has been explored by Brown, among others [Brow79] and is $\Theta(N)$. Although his method was not a visible surface algorithm, it could be adapted to this purpose. However, Brown treated the problem as one of geometry, not of graphics, and his solution is in *world space*. A visible surface algorithm for convex intersection that uses the frame buffer was presented in [FoFu86]. It is similar in many ways to the standard Z-buffer algorithm used for many polygon based systems. In the terms of [FoFu86], each pixel needs two registers. With a large frame buffer, providing enough bits for two registers is not too difficult as long as the stored values can be bounded.

At the beginning of the algorithm, in Pass 0, the value of *current_back* is set to the farthest possible value. This represents the background depth. The back-facing planes are scanned out first. If a plane is in front of the current farthest-forward back-facing plane, then that depth is stored for that pixel. For the sake of clarity the equation for *z* was used in the description of the algorithm, but in practice the the depth value is calculated incrementally at each pixel. Thus the calculation costs only one addition for each point.

The same process is followed for the front-facing planes. Each is scanned out incrementally and at each pixel the depth is compared to the current depth. If this plane is further back than the old one, then it becomes the current depth. However, if the plane is behind the most forward back-facing plane, then that pixel is not in the convex intersection. This is indicated by placing the same depth in both the current front and back registers. All the points at which this occurs are then set to the background colour in a quick Pass 3. Note that this third pass only scans the screen once, as did Pass 0.

The above procedure leaves on the screen the depth values for each visible point of the convex intersection. A fourth pass coloured the polyhedron for the

purpose of Figure 4.

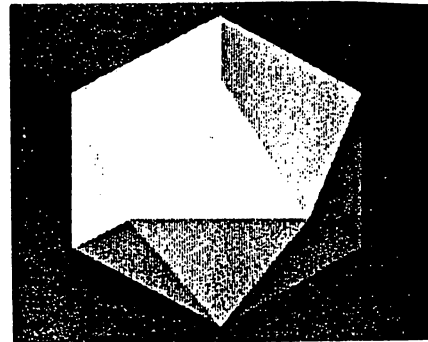


Figure 4 Example of Convex Intersection

There is one serious problem with the algorithm as defined: each plane must be scanned out across the entire screen. One can easily assume frame buffer hardware that accomplishes that in constant time. In fact this is very close to the algorithms used in *Pixel-Planes* [FGHS85]. As the current system was implemented with a general purpose graphics processor, this could be a limit on the performance of the algorithm. A way to avoid this extra work is evident from classic graphics algorithms. The polyhedron has some maximum and minimum *x* and *y* values on the screen. Simply "box" the polyhedron and only scan out the planes inside the box. Boxing the solid, however, leads to a new problem. The box is not quickly determined from a set of plane equations. The solution we adopted is to create the box dynamically. The first plane or two will be scanned out normally. By the third or fourth plane, there will be scanlines on which

Pass 0

For all pixels

current_back = MAXDEPTH

Pass 1

For each back-facing plane ($c > 0$)

$$z = -\frac{a}{c}x - \frac{b}{c}y - \frac{d}{c}$$

if $z < \text{current_back}$ then

current_back = *z*

Pass 2

For each front-facing plane ($c < 0$)

$$z = \frac{a}{c}x - \frac{b}{c}y - \frac{d}{c}$$

if $z > \text{current_front}$ then

if $z < \text{current_back}$ then

current_front = *z*

else

current_front = *current_back*

Pass 3

For all pixels

if *current_back* = *current_front* then

Colour = Background-colour

no part of the convex hull can possibly be. For example a back-facing plane could have cut in to a depth less than zero (i.e. behind the screen). In practice this eliminates a great many scanlines from consideration. The same process applies vertically. If the equations of the original planes are retained, then some beginning box can be computed from these. Since the number of initial planes is small (from 3 to 6) this is easy, and it has only to be done once. Then as the program runs, the box will be shrunk dynamically. The combination of the two boxing methods is quite efficient.

The half-space intersection algorithm then will take the output from the creation program to give a visible surface and depth values. The algorithm from [FoFu86] can be generalized to work on several convex hulls during the same run. The generalization only requires another register. This gives a total of three registers, which causes a problem for most frame buffers. As the entity stored represents depth, three registers in a 24-bit frame buffer means only 256 units of depth per register. This is not a great deal of room to work with. But it is only a temporary hardware limitation. We expect most future frame buffers to be more generous in bits/pixels. In fact there are already some with 48 bits, like the *Pixar* [LePo84].

The multiple convex intersection algorithm works very much like the single. The polyhedra are processed individually. This takes up the same two registers as before for *current_back* and *current_front*. The difference is that after a polyhedron is finished, it is then merged with those already scanned out. At each point, the depth of the just created surface is compared to the depth of the surface already there, if any. The surface closer to the viewer is kept in the third register. It should be noted that this algorithm not only allows multiple convex hulls, but that the hulls may actually intersect each other and the correct result will be obtained.

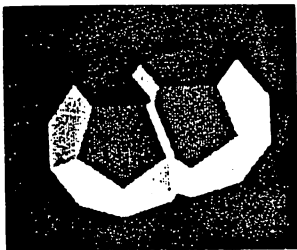


Figure 5 Example of Multiple Convex Intersection

The algorithms described so far results in an adaptive convex polyhedral shape to be written into the frame buffer for each tree. To make this shape more realistic, several methods can be used. One is to use stochastic interpolation [PiFo84, FoMi85] to "roughen" the hull by adding stochastic variations to the depth

of the visible faces. That will create rough (non-convex) edges, and possibly holes in the shape of the tree [Grin84]. On other is to use three-dimensional texture mapping. This is the technique that will be described in the next section, but it should be noted that they can and have been used concurrently.

3. Three-dimensional Texture Mapping

Texture mapping in two-dimension is a simple and powerful idea that has a long history in computer graphics [Catm74, BlNe76, Blin78]. More recently the idea was generalized to three dimensional texture [Perl85, Peac85][†]. What is needed is a texture solid, and a method to map it to the screen. The texture solid can be created by any process desired, as in the two dimensional case for the texture tile. The cube can be pre-computed, run-time computed, hand-drawn, or digitized from a real image. The creation is a process separate from the mapping. The mapping itself is simple in principle. The face of the object to be mapped has a set of coordinate values for its position. At each point on the object face, the (x,y,z) coordinate values are mapped into the (i,j,k) values of the texture cube.

One problem inherent to the idea of a three dimensional texture map is the sheer amount of storage necessary to hold the texture cube. One solution to the problem is to recognize that the frame buffer itself is a large block of memory. Assume a 32-bit frame buffer, not unreasonable by today's standards. This means that 32 bits of information are needed at each point in the cube. If the texture cube is stored in the top eight bits of each pixel, then four screen pixels store one texture pixel. Thus a 32 X 32 X 32 bit texture cube would take up $(2^5)^3 \cdot 2^2 = 2^{17}$ screen pixels. A 512 X 512 frame buffer contains 2^{18} points. It can be seen that even a sizable texture cube stored only in the top bits will easily fit into the frame buffer. By taking only the top eight bits, the lower 24 are left. Thus, the normal red, green and blue planes are untouched.

A second difficulty with three dimensional texture mapping is that of aliasing. This problem occurs, as it does in the two dimensional case, when a large scale difference between the texture cube and the object being mapped causes sampling problems. Solutions used in two-dimensional texture mapping can be applied here too. In particular the MIP map technique [Will83] directly translates to three dimensions. As in the two dimensional case the texture tile is repeatedly replicated at half resolution. Initially the texture cube takes up half of 512x512x8 bit buffer. If the cube is averaged into a cube half its length per side, it will only be one eighth of the size of the original. This process can be repeated and the eventual result will not even fill the buffer. This form of pre-computed averaging is a viable solution for at least some of the aliasing problems.

[†] The work described here was completed before these papers appeared, and thus our use of three-dimensional texture was developed independently. See [Grin84].

The fact that only the top of the frame buffer is used by the texture cube, has an important meaning. If the rest of the processing does not require the upper eight bits, then the texture cube can be pre-processed and read in before beginning the rest of the work. This means a considerable savings in run-time. Unfortunately, in this implementation the frame buffer contained only 24 bits per pixel. Since the half-space intersection algorithm needed 24 bits, this meant that the texture cube had to be read in only when it was to be used. This is yet another incentive to get as many bits in a frame buffer as you can afford. You will always find uses for them.

This method of storing the cube leads to a simple mapping function. Assume that a 2^n element-per-side cube is stored in a word-addressable 512 X 512 frame buffer. If the screen address is (x,y) with a depth value of z , then the mapping is a simple

$$\text{addr} = (x + y*2^n + z*2^{2n}) * 4.$$

In other words, the texture cube is treated as a large three-dimensional array. To find the exact (i,j) position in the frame buffer, the result above is split bit-wise. The lower 9 bits are the i position; the upper 8 bits are the j position. The multiplication by 4 is because 4 screen pixels store one texture point. Thus the mapping takes only three shifts and two additions per point.

The inputs (x,y,z) to the mapping function above must be contained in the cube. That is, with the above assumptions, $0 \leq x,y,z < 2^n$. To achieve this all that needs to be done is take the original (x,y,z) values modulo 2^n . This is equivalent to creating a large enough texture cube by repeating the smaller one over and over. It should be noted that because of the nature of the three dimensional cube, it is unlikely that there will be some undesirable macroscopic pattern created by this repetition, as often occurs in the two dimensional case. This is so, because in the two dimensional case, the same picture is repeated exactly. With a texture cube, this can only occur if the surface is at the same angle and position across several cubes, which is less likely to happen.

4. Mapping the Texture to the Polyhedra

In effect three-dimensional texture mapping allows the faces of the polyhedra we have defined previously to determine the boundaries of the tree in the *texture space*. The three dimensional texture cube can be generated by randomly placing small "chunks" of colour in three-space. The colours are chosen by the user, as well as the number of chunks and the percentage of each colour. It can also be generated using three-dimensional fractional Brownian motion [MaVN68, FoFC82], or other suitable procedural texture.

Each pixel which displays a part of the tree, contains three coordinates: the (x,y) position on the screen and the z value in the frame buffer. Each of these points is put through the inverse of the transformation applied to the objects to give the (x,y,z) real-world coordinates which will then be used as indices into the texture

cube as described above. This process give the consistency of colour desired and is also done with reasonable speed. The important point from the point of view of efficiency is that the mapping can be done incrementally.

The colouring of the tree is done in one pass through the screen. Each point is put through the inverse transform, and then mapped into the texture cube. At the top of the screen a current position in world space, (x_c, y_c, z_c) is calculated. This is obtained by putting the first point through the transform. Let the transformation be $\mathbf{M}: \mathbb{R}^3 \rightarrow \mathbb{R}^3$. Then for some Δx , since \mathbf{M} is linear,

$$\begin{aligned} \mathbf{M}(x + \Delta x, y, z) &= \mathbf{M}(x, y, z) + \mathbf{M}(\Delta x, 0, 0) \\ &= (x_c, y_c, z_c) + \Delta \mathbf{M}_x \end{aligned}$$

$\Delta \mathbf{M}_x$ is a constant for a constant Δx . This, of course, generalizes to $\Delta \mathbf{M}_y$ and $\Delta \mathbf{M}_z$. If the depth value changes non-linearly in the frame buffer, as it would if the tree has been stochastically "roughened", then an increment for a changing z value is needed. Again, the linearity of \mathbf{M} allows an incremental computation of $\mathbf{M}(x + \Delta x, y, z + \Delta z)$.

With this use of the three dimensional texture mapping, the tree has been coloured, with the ability to both reproduce the shading in place, and shade it correctly as the viewpoint moves. This all was accomplished with reasonable speed.

5. Adding the Trunk

Now that the crown of the tree has been shaped and shaded, the trunk of the tree is to be added. Part of the information stored during the processing of the three dimensional crown is the position of the centre of the tree. This is usually the base of one of the plane normals generated or given in the creation of the tree by splitting normals. After the centre position is put through its transforms, the resulting depth value lets a perspective mapping be done which scales the trunk to a size that fits the rest of the tree. This perspective mapping is a standard transform.

To shade the trunk, a modified version of Blinn's wrinkled surface technique was applied [Blin78]. The trunk is given a base colour, usually some dark brownish-red. Then ranges are given for each of the component colours (red, green, blue). A random amount within that range is added to the base colour at each point. For example, if the base colour is (60,30,10) and the ranges are (40,20,10), then the colour at each point of the trunk would be an r,g,b triple with red $\in (60,100)$, green $\in (30,50)$, and blue $\in (10,20)$. Values are uniformly distributed within these ranges. When the parameters were chosen well, this scheme gave a very acceptable simulation of tree bark. This method also lets different kinds of trees be modelled properly. Poplars, for example, have a smooth, light-green coloured bark, oaks a rough brown bark.

There remains to determine the visibility between crowns and trunks. One possibility is to use a reverse

painter's algorithm. The trunks are painted after all the crowns, from front to back, and they are not painted over anything already there. If the point of view is from above, such as every crown has priority over every trunk, this will give the correct priority.

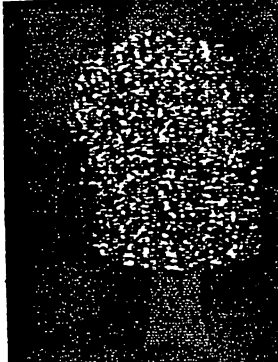


Figure 6 Completed Three Dimensional Tree

A more general method is to model the trunks as convex polyhedra, and use the algorithms applied to the crowns. The trunk can be described as a hexagonal cylinder, or cone, rendered with the half space intersection algorithm, and coloured as before. This approach gives an exact solution to the visibility problem, but adds seven or eight planes to scan out for each tree. It is not a large additional burden, especially since boxing is easier and more efficient given the shape of the trunk. It should be mentioned here that in our context we do not worry about modelling branches.

6. Implementation issues

We will describe in this section how the system was implemented, give numbers to indicate the system performance, and discuss ways in which this performance can be improved.

6.1. Implementation Description

The work of the system is split between two machines. The mainframe is a PDP VAX 11/780 running UNIX†. The other machine is an ADAGE RDS-3000 Graphics Processor and Raster Display System. This is a modular system with its own bus and it is interfaced to the VAX. The ADAGE bus is synchronous with a 32-bit data path. The basic cycle time is 200ns. The frame buffer is 512 by 512 pixels, each with 24 bits. It can also be organized in a 1K by 1K mode with 6 bits per pixel. Much of the power of the ADAGE comes from the use of the 200ns cycle, 32 bit, bit-slice processor. The processor is supported by a

† UNIX is a trademark of Bell Laboratories

4K by 64-bit wide microcode memory and an 8K by 32-bit wide scratchpad memory. The processor also includes a 16 X 16 bit hardware multiplier which does a signed multiplication in two cycles (400ns). The code for the graphics processor was written in a C-like language for a compiler developed at the University of North Carolina [Bish82]. While allowing only integer arithmetic, this language was of immeasurable help to the implementation.

Almost all of the actual processing work for the tree creation system implemented was done on the ADAGE bit-slice processor (herein called simply the Adage). The VAX processor was used only as a driver, loading microcode into the Adage and starting the routines, and to perform the basic geometric operations of splitting the normals.

In some parts of the system it was necessary to do non-integer arithmetic on the Adage. The best example of this was the convex intersection routine. A series of fixed point routines (one 16 bit word for the integer part and one 16 bit word for the fraction) were implemented. In addition to needing non-integer arithmetic, several of the Adage routines needed random, or at least pseudo-random, numbers. We used a multiplicative congruential routine to generate the pseudo-random numbers. To ensure that the routine did not loop, a new random seed was used every 512 iterations. This method did not consume excessively large amounts of time to feed seeds down to the Adage but did generate satisfactory random numbers for the Adage routines.

6.2. System performance

Detailed timing information can be found in [Grin84]. For the icosahedron of Figure 4, with an initial box of 512x512, the rendering takes roughly 12 seconds. These assumptions give a time of approximately 50μsec per pixel, or about .7 to .8 seconds per plane.

The roughening step, if applied takes about 1.0 second for a 250x250 pixel object. The texture generation, takes also about one second, but again this is a preprocessing step if sufficient storage is available for the texture.

The other important factor is the time to load each separate program in the processor, when the micro-store is not big enough, which was the case in our system. This is also dependent on the load on the VAX and can take several seconds.

6.3. Possible Speed Improvements

At present the tree creation system is several orders of magnitude away from being real-time. The key to improve the performance is in a combination of specialized processors and a suitable multiprocessor architecture.

Specialized processors already exist for the type of operations used in the system. For the creation of the planes by normal splitting, most of the operations are floating point operations, with calls to a normal distribution function, and to trigonometric functions. The functions can be replaced by lookup tables. In this case, each splitting operation takes less than 20

floating point operations and/or lookup steps. The number of splits necessary depends on the number of trees, and their size on screen. It also depends on how many different trees the system uses. There can be many trees on the picture sharing the same convex polyhedron. To take a numerical example, assume a 512x512 display, 200 trees, each on the average 20x20 pixels, and covering 1/4 of the screen, that is an average depth complexity (for the trees only) of 1.22. Further assume that a tree on the average goes from 6 planes (in the initial master) to 12 on the picture, that is needs 6 plane splitting operations. At 60 frames/second, that means 1.4 MFLOPS for the processor in charge of the splitting. This is easily achievable on a custom VLSI.

The second step, and the main bottle-neck in the current system, is the computation of the convex intersection. As mentioned before, an architecture such as used in the Pixel-planes is suitable for the basic operations used in this step. Making the assumptions in [FGHS85], that is a 10Mhz clock, and reasonable values for the number of bits in the plane equations, we obtain about 60 clock cycles per plane scanned out, that is each plane is scanned out in 6 μ s. The trees can then be scanned out in 14 ms, which is fast enough. Note that this is independent of the size of the trees.

The stochastic values needed for the roughening step and the three-dimensional texture generation can be supplied by a processor like the STINT [PiFo84]. The current implementation of the STINT generates two-dimensional texture, and can only generate a 70x70 texture in real time, but most of the textures needed can be precomputed. It also should be noted that specialized hardware for real time texture mapping is already in use in flight simulators such as Evans & Sutherland CT6 or General Electric Compuscene.

Remains to organize these processors into a suitable display architecture. This is a complex task, especially since there are other parts of the display system to consider (terrain, buildings, moving vehicles, atmospheric effects, etc.). This is left, as they say, to further research.

7. Conclusions

Within the stated limits: reasonably realistic trees, simple operations, adaptability and flexibility, we feel that the techniques described here succeeded fairly well. One interesting lesson is also that the system distinguishes clearly between the modelling of the shape, which is done with the implicit intersection of half-spaces, and the rendering method, which is the combination of a frame buffer algorithm and three-dimensional texture mapping.

We saw also that the simplicity of the operations and their modularity led to the conclusion that with suitable specialized processors, the real-time generation and display of several hundreds such trees is possible.

Acknowledgements

We gratefully acknowledge the support of the Natural Sciences and Engineering Research Council of Canada. Alain Fournier also wants to thank the Department of Computer Science of Stanford University for its hospitality and the use of its facilities to write this paper.

References

- [AoKu84] Aono, M. and Kunii, T. L., "Botanical Tree Image Generation", IEEE Computer Graphics and Applications, 4, 5, (May 1984), 10-34.
- [Bish82] Bishop, G., *Gary's Ikonas Assembler Version 2 Differences Between Gia2 and C*, Technical Report, University of Northern Carolina, 1982.
- [BIne76] Blinn, J., and M. E. Newell, "Texture and Reflection in Computer Generated Images", Communications of the ACM, 19, 10, (Oct. 1976), 542-547.
- [Blin78] Blinn, J. F., "Simulation of Wrinkled Surfaces", in *Proceedings of SIGGRAPH'78*, also published as Computer Graphics, 12, 3, (Aug. 1978), 286-292.
- [Bloo85] Bloomenthal, J., "Modeling the Mighty Maple", in *Proceedings of SIGGRAPH'85*, also published as Computer Graphics, 19, 3, (July 1985), 305-311..
- [Brow79] Brown, K. Q., *Geometric Transforms for Fast Geometric Algorithms*, Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, 1979.
- [Catm74] Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces*, University of Utah Computer Science Dept., UTEC-CSc-74-133, (Dec. 1974).
- [FGHS85] Fuchs, H., Goldfeather, J., Hultquist, J. P., Spach, S., Austin, J. D., Brooks, F. P., Eyles, J. G. and Poulton, J., "Fast Spheres, Shadows, Textures, Transparencies and Image Enhancements in Pixel-planes", in *Proceedings of SIGGRAPH'85*, also published as Computer Graphics, 19, 3, (July 1985), 111-120.
- [FoFC82] Fournier, A., Fussell, D. and Carpenter, L. "Computer Rendering of Stochastic Models", Comm. ACM, 25, 6, (June 1982), 371-384.
- [FoFu86] Fournier, A. and Fussell, D., "On the Power of the Frame Buffer", to appear in ACM Transactions on Graphics.
- [FoMi85] Fournier, A. and Milligan, T., "Frame Buffer Algorithms for Stochastic Models", IEEE Computer Graphics and Applications, 5, 10, (October 1985), 40-46.
- [Gard84] Gardner, G. Y., "Simulation of Natural Scenes Using Textured Quadric Surfaces",

- in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 11-20.
- [Gard85] Gardner, G. Y., "Visual Simulation of Clouds", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 297-303.
- [Grin84] Grindal, D. A., *The Stochastic Creation of Tree Images*", Master Thesis, Department of Computer Science, University of Toronto, (April 1984).
- [LePo84] Levinthal, A. and Porter, T., "Chap, a SIMD Graphics Porcessor", in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 77-82.
- [MaVN68] Mandelbrot, B. B. and Van Ness, J. W., "Fractional Brownian Motion, Fractional Noises and Applications", *SIAM Review*, 10, 4, (October 1968), 422-437.
- [Peac85] Peachy, D. R., "Solid Texturing of Complex Surfaces", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 279-286.
- [Perl85] Perlin, K., "An Image Synthetizer", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 287-296.
- [Pipe84] Piper, T. and A. Fournier, "A Hardware Stochastic Interpolator for Raster Displays", in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 83-91.
- [ReBl85] Reeves, W. T. and Blau, R., "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", in *Proceedings of SIGGRAPH'85*, also published as *Computer Graphics*, 19, 3, (July 1985), 313-322.
- [Reev83] Reeves, W., "Particle Systems - A Technique for Modelling a Class of Fuzzy Objects", in *Proceedings of SIGGRAPH'83*, also published as *Computer Graphics* 17, 3, (July 1983), 359-376
- [Smit84] Smith, A. R., "Plants, Fractals and Formal Languages", in *Proceedings of SIGGRAPH'84*, also published as *Computer Graphics*, 18, 3, (July 1984), 1-10.
- [Will83] Williams, L., "Pyramidal Parametrics", in *Proceedings of SIGGRAPH'83*, also published as *Computer Graphics*, 17, 3, (July 1983), 1-11.

METHODS FOR STOCHASTIC SPECTRAL SYNTHESIS

j.p. lewis

Computer Graphics Laboratory
New York Institute of Technology

ABSTRACT

A number of spectral modeling approaches in the engineering and estimation literature are potentially applicable to stochastic synthesis in computer graphics. Two specific approaches are developed. The orthogonality principle of estimation theory is used to derive a stochastic subdivision construction with specified autocorrelation and spectrum properties; this approach also provides an alternative theoretical basis for the popular fractal subdivision algorithms. A shaped Poisson point process is a second approach which conveniently separates the spectral and graphic modeling problems. Synthetic textures and terrains are presented as a means of visually evaluating the constructed noises.

KEYWORDS: stochastic models, texture synthesis, fractals, terrain modeling.

RESUME

Les méthodes de modélisation spectrales empruntées aux sciences de l'ingénieur, ou dérivées de la théorie de l'estimation peuvent être appliquées à la synthèse stochastique dans le champ de l'informatique graphique. Deux points de vues sont présentés dans cette communication. A partir du principe d'orthogonalité de la théorie de l'estimation on peut dériver une méthode de subdivision stochastique possédant certaines spécifications d'autocorrélation et propriétés spectrales; cette approche fournit aussi une base théorique nouvelle pour la construction d'algorithmes de subdivision fractale. Un processus utilisant un filtrage de l'impulsion de Poisson fournit une deuxième approche, qui permet de déterminer une séparation claire des problèmes de nature spectrale de ceux liés à la modélisation graphique. Les textures synthétiques et les modèles de terrains présentés permettent d'évaluer visuellement les bruits ainsi générés.

MOTS CLEFS: Modèles stochastiques, Textures synthétiques, fractal, modélisation de terrain.

1. Introduction

Stochastic techniques have assumed a prominent role in the synthesis of complex and naturalistic imagery, for example, [1][2][3][4][5][6][7]. This role has been termed *amplification* [5]: the image modeler specifies a pseudo-random procedure and its parameters; the procedure can then automatically generate the vast amount of detail necessary to create a realistically complex scene. The success of stochastic modeling depends both on its economy and on our ability to construct stochastic models to approximately emulate a variety of phenomena. The full power of stochastic modeling has not been achieved in existing techniques. For example, the widely-used stochastic fractal techniques model only spectra of the form f^{-d} , and thus cannot describe phenomena with scale-dependent detail or directional or oscillatory characteristics.

The problem of modeling a random process ("noise") with an arbitrary spectrum is well understood. Basically, the procedure is to filter an uncorrelated noise (as obtained from a random number generator) to obtain the desired spectrum. The spectrum of the filtered noise is simply the squared magnitude of the transfer function of the filter. Using this synthesis procedure, many of the filtering and spectral analysis approaches described in the literature are potentially applicable to the problem of stochastic modeling in computer graphics. This paper adopts two approaches, optimal mean-square estimation and a shaped point process model, to produce stochastic synthesis algorithms which are computationally suitable for computer graphics applications.

2. Generalized Stochastic Subdivision

The stochastic subdivision construction described by Fournier et. al. [1] may be generalized to synthesize a noise with an arbitrary prescribed spectrum (the generalized subdivision technique is described in more detail in [8]). The basis of the Fournier et. al. construction is a midpoint estimation problem: given two samples considered to be on the noise, a new sample midway between the two is estimated as the mean of the two

samples, plus a random deviation whose variance is the single noise parameter. The construction is based on two properties of fractional Gaussian noise:

- 1) When the values of the noise at two points are known, the expected value of the noise midway between two known points is the average of the two values.
- 2) The increments of fractional Gaussian noise are Gaussian, with variance which depends on the lag and on the noise parameter.

Since only the immediately neighboring points are considered in making the midpoint estimation, the noise autocorrelation information is not used, and the constructed noise is Markovian. This is not a limitation as long as the construction is used as an approximate (stationary) model for Brownian motion. However, the construction has been applied to the non-Markovian fractional noises f^{-d} , $d \neq 2$ [9]; in these cases, disregarding the autocorrelation produces "creases".

The general problem of estimating the value of a stochastic process given knowledge of the process at other points is the subject of estimation theory and of the Wiener and Kalman filtering techniques [10]. The *orthogonality principle* indicates that the mean-square error of a stationary linear estimator will be minimum when the error is orthogonal in expectation to the known values on the process. It is also known that when the estimated process is Gaussian (as in the case of fractional noises), the linear estimate is optimal in the sense of being identical to the best nonlinear estimate given the same number of observations [11][12]. Stochastic subdivision is specifically similar to the application of digital Wiener filtering in the linear-predictive coding (LPC) of speech [13], since in both of these applications points on a stochastic process are estimated, and then perturbed and re-used as "observations".

In our case, the midpoint \hat{x} at each stage in the construction will be estimated as a weighted sum of the noise values x known from the previous stages of the construction, in some practical neighborhood of size $2S$:

$$\hat{x}_{t+0.5} = \sum_{k=1-S}^S a_k x_{t+k}$$

(with t indexing the points known at the previous construction stage). The estimated value $\hat{x}_{t+0.5}$ will form a new noise point with the addition of a random number of known variance; the new points will in turn form some of the data in subsequent construction stages.

The orthogonality principle then takes the form

$$E \left\{ x_{t+m} \left(x_{t+0.5} - \sum_{k=1-S}^S a_k x_{t+k} \right) \right\} = 0$$

or

$$E \left\{ x_{t+m} x_{t+0.5} - \sum_{k=1-S}^S a_k x_{t+m} x_{t+k} \right\} = 0$$

for $1-S \leq m \leq S$. Recalling that the expectation of $x_{t+i} x_{t+j}$ is the value $R(i-j)$ of the noise autocorrelation function R (for a stationary noise), we obtain the equation

$$R(m-0.5) = \sum_{k=1-S}^S a_k R(m-k)$$

which can be solved for the coefficients a_k given R . The matrix $R(m-k)$ is Toeplitz, permitting the use of efficient algorithms available for the inversion of these matrices, such as the Levinson recursion [14]. The mean-square estimation error

$$E\{(x - \hat{x})^2\} = R(0) - \sum_{k=1-S}^S a_k R(0.5-k)$$

is used to select the noise variance (and optionally the neighborhood size) at each construction stage [8]. Fig. 1 illustrates successive stages in generalized subdivision to an oscillatory noise with an autocorrelation $R(\tau) = \cos(\omega\tau) \exp(-\tau^2)$.

2.1. Subdivision in two dimensions

The significant difference from the one-dimensional solution is that there are now several classes of points to be estimated, categorized by their spatial relationship to the points computed at previous subdivision levels (this depends on the selected interpolation mesh). For the planar quadrilateral mesh shown in Fig. 2 the mid-face vertex 'x' will require different coefficients than the mid-edge vertices 'o'. For example, (using our coordinate system with the midpoints "indexed" by $1/2$) the midpoint coefficients are obtained by solving

$$\begin{aligned} R(j-0.5, i-0.5) \\ = \sum_{r=1-S}^S \sum_{c=1-S}^S a_{r,c} R(j-r, i-c). \end{aligned}$$

for $1-S \leq j, i \leq S$. This equation can be considered as a system $Ax = b$ by rewriting $R(y, x)$ and $a_{r,c}$ as vectors by a consistent ordering of the subscripts; the dimension of the matrix A is now the square of the neighborhood size $2S$.

2.2. Evaluation

The generalized subdivision technique produces high-quality noises with specified spectra and eliminates the creases associated with stochastic subdivision to non-Markovian noises. It also shares the attractive properties of the stochastic subdivision construction [1], i.e., the consistency properties described in [1] including the ability to model a noise at different resolutions, and the ability to model regions of a noise in any order (a "non-causal" property which is not available in Fourier synthesis and other spectral synthesis approaches). When a separable Markovian autocorrelation function $R(x, y) = \exp(-|x|) \exp(-|y|)$ is specified, the generalized subdivision reduces to a form of fractal subdivision, in the sense that only the coefficients for the nearest neighbors of an estimated midpoint are non-zero. Subdivision to non-Markovian spectra is computationally more

expensive due to the larger neighborhood sizes required. Fig. 3 shows several textures produced with the generalized subdivision technique and Figs. 4, 5 illustrate two height fields produced using this technique, displayed as synthetic terrains.

Several limitations of the generalized subdivision technique are:

One must know or invent the noise autocorrelation function. Since the autocorrelation function is the Fourier transform of the power spectrum (Wiener-Khinchine relation), and the latter must be non-negative, the autocorrelation function must be *non-negative definite*. Unless this constraint is well understood, it may be easier to design the power spectrum and obtain the autocorrelation by transformation, or to restrict one's choice to paradigmatic or empirically estimated autocorrelation functions.

A second restriction of the generalized subdivision technique derives from the variable-resolution property of subdivision constructions. The identification of different stages in the construction with different resolutions is strictly incorrect. This can be seen from one point of view by considering the problem of obtaining a half-resolution version of a given noise record. A half-resolution noise which preserves the spectral content of the original up to the new, lower Nyquist rate is achieved by low-pass filtering, followed by dropping every other sample ("decimation"). The half-resolution noise resulting from reducing the recursion level in a stochastic subdivision construction is achieved by decimating without filtering. A half-resolution noise does not in general coincide with every other sample of the original noise unless the latter has no detail at frequencies above half its Nyquist rate. Thus, any spectral energy above half the original Nyquist rate is aliased in changing the resolution through the subdivision construction depth.

Significantly, an aliased noise does not form coherent artifacts such as Moire patterns; rather, the noise at the lower resolution appears as a somewhat different noise than the original, so the subject may appear to "bubble" during a zoom. The aliasing is limited for noises with monotonically decreasing spectra such as fractal noises, since the majority of the spectral energy remains unaliased in any resolution change. However, serious anomalies may occur if the resolution of a noise whose spectrum is flat or increasing at some frequencies (as may be achieved with the generalized subdivision technique) is varied by changing the subdivision recursion depth.

3. Shaped Point Process

A second stochastic synthesis algorithm is suitable when samples of the desired noise are available. An analysis-synthesis approach would analyze the noise to determine parameters of a stochastic model, and then apply the model to generate a synthetic noise. If the only goal is to synthesize the noise, however, a more direct approach is

feasible: the noise x is produced by a (discrete) convolution

$$x_t = \sum_{k=-S}^S h_k u_{t-k}$$

of an uncorrelated noise u with the (windowed) noise sample h of size $2S+1$, with h playing the role of a filter kernel. The autocorrelation of x is easily derived:

$$\begin{aligned} R(\tau) &= E\{x_t x_{t+\tau}\} \\ &= E \sum_k \sum_m h_k h_m u_{t-k} u_{t+\tau-m} \end{aligned}$$

The noise u is stationary and uncorrelated so the expectation of the factors $u_{t-k} u_{t+\tau-m}$ is $E\{u^2\} \delta(\tau+k-m)$, so

$$R(\tau) = \sum_k h_k h_{k+\tau} \quad (1)$$

The power spectrum of x is the Fourier transform of R . Since (1) is a convolution $h_t * h_{-t}$, its transform is (by the convolution theorem [15])

$$S(\omega) = H(e^{j\omega})H(e^{-j\omega}) = |H(e^{j\omega})|^2$$

so the spectrum of x is that of h (as expected). The spectrum of the noise sample h will in turn resemble that of the prototype noise if it is large enough to include any low-frequency components characteristic of the prototype and if it is windowed to reduce the effects of discontinuities at the sample boundary.

Convolution with a large noise sample is inefficient and the convolution would usually be implemented in the frequency domain by FFT. Computational economy can also be achieved by replacing the noise u with a 'sparse noise' or particle system (sampled Poisson point process) \hat{u} which is non-zero at a limited number of points under the sample h . The reduced convolution takes the form

$$x_t = \sum_k \hat{u}_k h(t-t_k) \quad (2)$$

where t_k is the location of the k th non-zero point of the process, and the summation is now over these points rather than over h (a similar technique was described as one of the methods in [16] but its use as a general spectral modeling approach was not fully developed there). The autocorrelation and spectrum are unchanged provided the values of \hat{u} are independent. This "shaped point process" resembles both shot noise (in which the noise \hat{u} is defined to be a constant-amplitude Poisson impulse process), and a generalized form of pulse amplitude modulation reconstruction, which requires t_k to be evenly spaced.

3.1. Spectral and graphic modeling

The primary advantage of this algorithm is not efficiency, however, but that it suggests manipulating the point process as an entity itself. For example, to produce a 'fluid texture' by animating the point process requires only updating the location of each point by a dynamic equation, whereas manipulating a uniformly sampled noise

field to the same effect requires computations more analogous to those of a fluid flow problem on a uniform grid. Similarly, points may be restricted to an area of the plane with conceptually simple algorithms such as Monte Carlo or an ad hoc placement procedure, whereas restricting a noise field requires scan converting the boundary of the region or a global windowing operation.

The non-causal property of subdivision methods is achieved in a shaped point process using an appropriate (non-causal and consistent) construction of the point process. A simple construction is to divide the noise domain into numbered cells and approximate the Poisson point process by N points in each cell, with the random number generator seeded by the cell number. The value of the shaped noise at a particular point is obtained by (2) summed over only the points in those cells which are closer than a radius the size of the kernel.

The kernel h can also be manipulated independently of the point process. The spectral bandwidth of a shaped point process is entirely determined by the kernel. If the size of the kernel is small compared to the depth in a perspective view of a shaped point process noise, the noise can be accurately and efficiently anti-aliased by selecting appropriate precomputed bandlimited versions of the kernel as a function of depth. The kernel can be varied as a function of the position of each point to produce a *non-stationary* noise. For example, wind-blown clouds or terrain ridges where the directional tendency varies over the scene could be emulated by rotating the kernel as a function of position. This type of control is not directly available in most filtering techniques; e.g. it is achieved in a Fourier transform method only by breaking the noise into small overlapping stationary regions and interpolating the synthesis on these regions (overlap-add method for short-time Fourier transformation).

Thus, a shaped point process provides a convenient separation between the spectral modeling problem (obtaining the kernel) and the graphic modeling problem of shaping the noise to form a subject. (A similar separation occurs in 'waveform' speech synthesis: a kernel is used to model the formant (spectral) shape; it is convolved with a impulse sequence or noise representing the voice pitch and amplitude [17]).

3.2. Evaluation

The shaped point process is a simple means of approximately "resynthesizing" noises. The method also generalizes directly to several dimensions. Fig. 6 shows the shaped point process resynthesis of several texture samples from the Brodatz album [18]. Resynthesis is of course more intuitive than specifying the parameters of a texture model, and it allows the generation of homogeneous noises of arbitrary extent. Periodic noises can be produced by altering the addressing in (2) to wrap around specified boundaries; this is a useful property in applications such as texture mapping. The shaped point process can also be applied with an analytically defined kernel;

the lower right plot in Fig. 6 is a perspective view of a wave-like texture created with a bandpass kernel of the form $R(x, y) = \cos(\alpha x + \beta y) \exp(-\sqrt{x^2 + y^2})$.

The textures in Fig. 6 also suggest the limitations of the shaped point process method, and of spectral methods in general. The phase spectrum in a spectral synthesis method is that of the driving noise, which is random. Thus spectral synthesis cannot produce a coherent-phase texture such as a brick wall pattern. In fact, given a step function for the kernel h , the shaped impulse process will result in a f^{-2} noise -- the spectrum of the kernel is reproduced but the visual appearance is quite different.

The grey levels in a texture photograph reflect the illumination of the texture and may not directly correspond to 'physical' properties of the texture such as color or relief depth. Thus, a texture synthesized from a photographic sample will reproduce the spectral character of the texture as illuminated rather than as we perceive it. One common effect is that sharp cast shadows produce discontinuities in the texture kernel and so introduce f^{-2} noise into the synthesized texture.

Unlike subdivision constructions, a shaped point process noise has definite inner and outer scales. The autocorrelation (1) is zero beyond the width of the kernel, so the noise is uncorrelated at scales larger than this width (this can be seen in Fig. 6 as the scale at which the textures become "blotchy"). The inner scale is of course the Nyquist rate determined by the (fixed) sample rate of the noise. The bandwidth available in a shaped point process is nevertheless considerably greater than that available in many artificial texturing methods (e.g. [19]) and is adequate for many purposes, since a stochastic model will rarely be applicable over a very broad range of scales in any case. Also, some phenomena such as waves, fire, and bark which might be modeled by stochastic methods are often fairly smooth above and below a range of scales.

4. Non-Gaussian Noises

By a loose version of the central limit theorem, the probability density of a noise produced with spectral synthesis will tend to be Gaussian regardless of the density of the driving noise, since the spectral shaping operation is effectively a linear filter or a weighted sum of the input noise values [15]. It is sometimes desirable to model non-Gaussian processes. For example, with respect to the uniform or normal distributions, a distribution such as $\exp(-|x|)$ has an increased number of 'events' far removed from the mean. Transforming a Gaussian noise to have a higher-variance non-Gaussian distribution tends to differentially exaggerate the most pronounced portions of the noise and so can produce the impression of a 'subject' against a background, or of a non-stationary noise. Some of the published fractal landscape pictures depict fractional noises passed through a square or cube non-linearity which improves their appearance.

The probability density of a random process can be shaped by means of a memoryless nonlinear transformation $g(x)$. For this purpose it is sufficient to consider only monotonically increasing $g(x)$. Then, by "conservation of probability", the probability of an event $y < y$ where $y = g(x)$ is identical to that of the event $x < x$:

$$F_y(g(x)) \equiv P\{y < y\}$$

$$= F_x(x) \equiv P\{x < x\}$$

or

$$f_y(y) = f_x(g^{-1}(y)) \frac{dg^{-1}(y)}{dy}$$

so

$$g(x) = F_y^{-1} [F_x(x)]$$

Two cases are particularly useful. When x is uniformly distributed in $(0,1)$, $F_x(x) = x$ so the nonlinear function $g(x)$ which shapes a uniform noise x to have a desired distribution F_y is just $g = F_y^{-1}$. When the desired distribution F_y is uniform, $g = F_x$. Thus, the procedure to transform a noise to have a desired distribution is to first pass the noise through its own distribution function to make a uniform $(0,1)$ noise, and then use the result to index the inverse of the desired distribution function. Both of these operations can be implemented by table lookup for reasonably smooth functions, so distribution shaping can be very efficient.

4.1. Effect on spectrum

The nonlinearity which shapes the distribution can also have a powerful effect on the spectrum of a correlated noise, however. This can be appreciated by considering the potential effect of a nonlinearity $g(x)$ on a single "frequency component" $\cos(\omega t)$. By choosing $g(x) = f(\cos^{-1}(x))$, an arbitrary periodic waveform $f(\omega t)$ is produced at the output of the nonlinearity given the single frequency as input. The envelope of the spectrum at the output of the nonlinearity also depends on the amplitude of the input signal. A signal passed through a nonlinearity does not obey either the superposition or homogeneity principles of linear systems, so the effect of a nonlinearity on a noise cannot be analyzed as the superposition of its frequency components.

A general expression for the autocorrelation function at the output of $g(x)$ is [20]

$$R(\tau) = \iint g(x_1)g(x_2)f_x(x_1, x_2, \tau)dx_1dx_2$$

where f_x is the second-order joint probability density of the input. The spectrum of the output is the transform of this. However, this integral is difficult to evaluate and analytic solutions are known only for some special cases, including various cases where f_x is Gaussian. Beckmann [20] gives an expression for the distorted correlation function of a Gaussian noise as a series involving weighted powers of the input autocovariance. The output spectrum is the transform of this series, which by the modulation (or convolution) theorem is a weighted series of n -th-order self convolutions of the input spectrum. This

effect is illustrated in Fig. 7. In theory it should be possible to design the spectrum of the undistorted noise so that a desired spectrum is achieved *after* distortion, but this approach has not been formulated to the author's knowledge.

We conclude that nonlinear distortion is a powerful means of generating correlated non-Gaussian noises. However, this approach should be used carefully if accurate control of the spectrum and probability density are required. For example, some of the "fractal Gaussian" terrains we have seen are probably neither Gaussian nor of the attributed spectral exponent or fractal dimension as a result of squaring or other nonlinear distortions (e.g. a squared Gaussian noise has a one-sided probability density

$$f_y(y) = \frac{1}{\sqrt{2\pi y}} e^{-y/2}, y \geq 0$$

which is quite different from the Gaussian density).

5. Conclusion

Two spectral methods for stochastic synthesis were described. Spectral approaches allow the synthesis of noises with arbitrary power spectra, and so can describe both narrowband deterministic-like noises such as [19] and broadband random noises such as fractals, as well as noises which exhibit a mixture of structure and randomness.

References

1. Fournier, A., Fussell, D., and Carpenter, L., Computer rendering of stochastic models, *Communications ACM* 25, 6, June 1982, 371-384.
2. Kawaguchi, Y., A morphological study of the forms of nature, *Siggraph 82 Proceedings* July 1982.
3. Reeves, W., Particle Systems--A Technique for Modeling a Class of Fuzzy Objects, *ACM Trans. Graphics* 2, 2, April 1983, 91-108.
4. Reeves, W. and Blau, R., Approximate and probabilistic algorithms for shading and rendering structured particle systems, *Siggraph 85 Proceedings* July 1985, 313-322.
5. Smith, A. R., Plants, Fractals, and Formal Languages, *Computer Graphics* 18, 3, July 1984, 1-10.
6. Voss, R., Random fractal forgeries, *Siggraph conference tutorial notes* July 1985.
7. Perlin, K., An image synthesizer, *Siggraph 85 Proceedings* July, 1985, 287-296.
8. Lewis, J.P., Generalized stochastic subdivision, *submitted for publication*. Jan. 1986.
9. Fournier, A. and Milligan, T., Frame buffer algorithms for stochastic models, *Proceedings, Graphics Interface* May 1985, 9-18.
10. Deutsch, R., *Estimation Theory*, Prentice-Hall, New Jersey, 1965.
11. Yaglom, A., *An Introduction to the Theory of Stationary Random Functions*, Dover, New York, 1973.

12. Papoulis, A., *Signal Analysis*, McGraw Hill, New York, 1977.
13. Markel, J. and Gray, A., *Linear Prediction of Speech*, Springer-Verlag, New York, 1976.
14. Levinson, N., The Wiener RMS (root mean square) error criterion in filter design and prediction, *J. Math. Phys.* **25**, 1947, 261-278.
15. Bracewell, R., *The Fourier Transform and Its Applications*, McGraw-Hill, New York, 1965.
16. Lewis, J.P., Texture synthesis for digital painting, *Computer Graphics* **18**, 3, July 1984, 245-252.
17. Baumwolsper, M., Speech generation through waveform synthesis, *IEEE Acoustics, Speech and Signal Processing Conference* 1978.
18. Brodatz, P., *Textures: A Photographic Album*, Dover, New York, 1966.
19. Schachter, B., Long Crested Wave Models, *Computer Graphics and Image Processing* **12**, 1980, 187-201.
20. Beckmann, P., *Probability in Communication Engineering*, Harcourt-Brace, New York, 1967.
21. Vanmarcke, E., *Random Fields*, MIT Press, Cambridge, 1983.

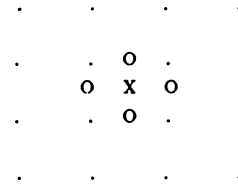


Fig. 2 : Planar quadrilateral subdivision mesh using a 4^2 neighborhood. The vertices 'o' and 'x' are estimated using the surrounding 'observation' points '.'.

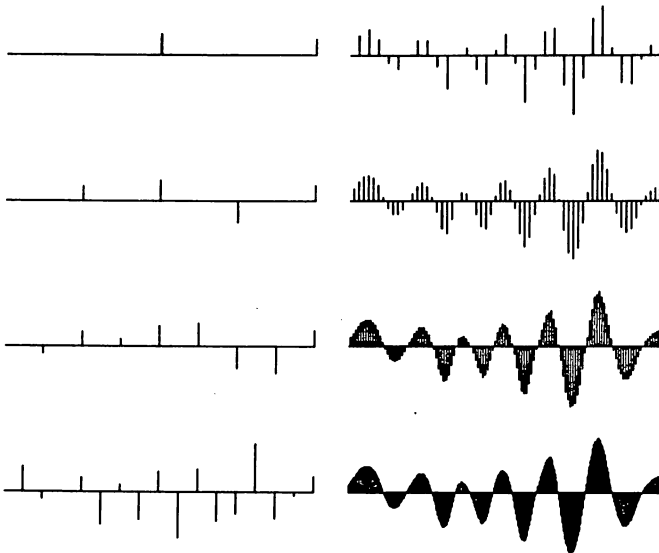


Fig. 1: (top to bottom) Stages in generalized subdivision to a non-fractal (oscillatory) noise.

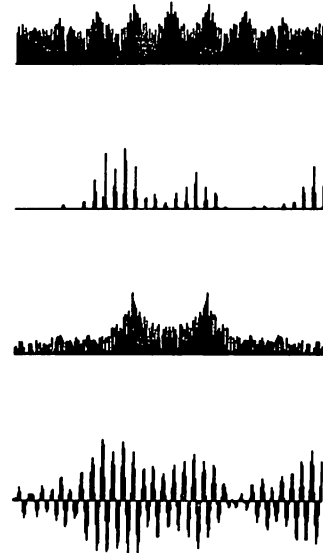


Fig. 7 : Bandpass noise and spectrum (lower figures) and noise and spectrum at the output of a pair of nonlinearities effecting a hyperbolic probability density. The self-convolution of the input spectrum produced by the nonlinearities results in an odd-harmonic spectrum structure.

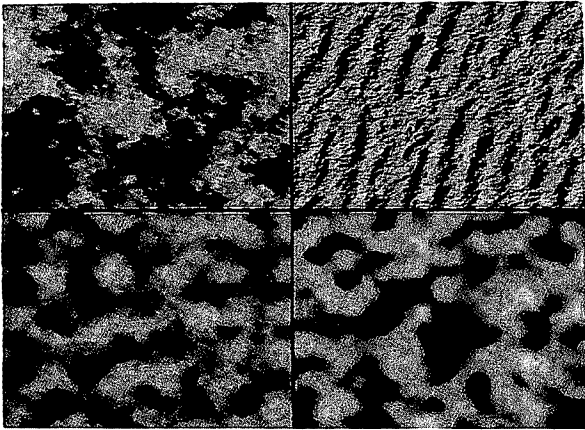


Fig. 3 : Several textures produced using the generalized subdivision technique. Clockwise from top left: Markovian, oscillatory (shaded as an obliquely illuminated height field), Gaussian, and highpass isotropically oscillatory textures.

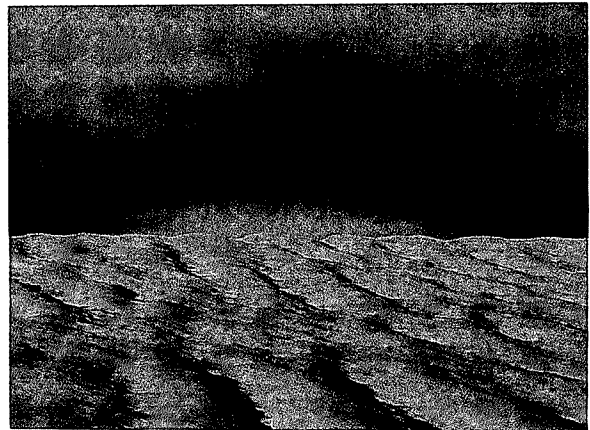


Fig. 5 : Synthetic sky and terrain with directional trend produced with generalized subdivision.

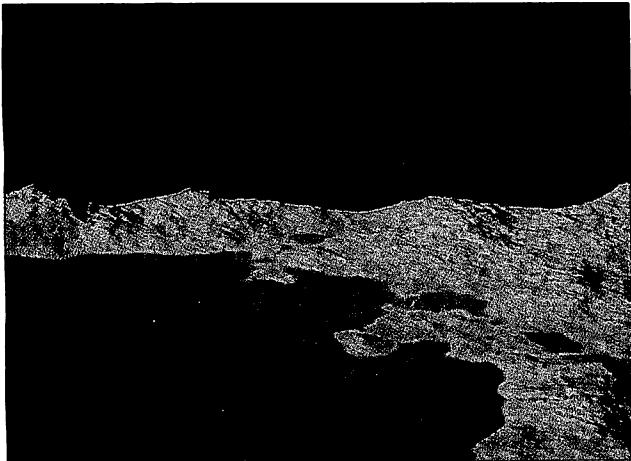


Fig. 4 : Generalized subdivision terrain with an isotropic autocorrelation $R(x, y) = \exp(-(x^2 + y^2)^{0.7})$. This figure resembles a power-modified fractal terrain but it can be distinguished (in being smoother) in a comparison.

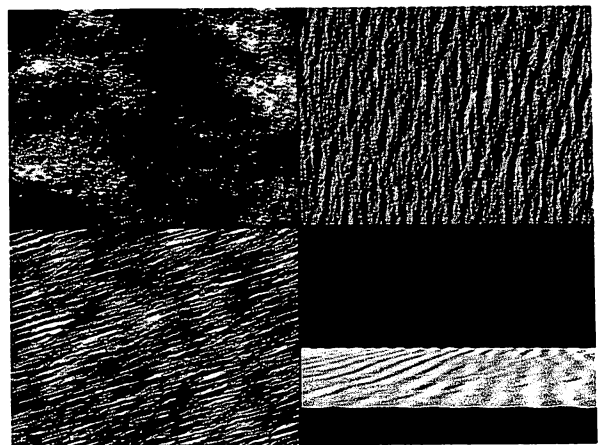


Fig. 6 : Several shaped impulse process textures. Counter-clockwise from top right: rough waves (shaded as an obliquely illuminated height field), fieldstone, and straw synthesized from Brodatz [18]. The last figure is a perspective view of a wave-like texture.

Interactive 3-D Modeling with Personal Computers

Robert W. Thornton and Gregory J. Glass
New York Institute of Technology
Computer Graphics Laboratory
Old Westbury, New York 11568

Abstract

An interactive three-dimensional modeling system, called *Facet*, is presented. *Facet* was developed for use during the design of physical artifacts: from studies of simple shapes, spaces, and forms to detailed modeling of complex physical systems. *Facet* provides a visual medium for enhancing perception of the model and for exploring various alternatives during the design process. In order to make this 3-D modeling medium available and affordable to any interested design professional *Facet* operates on commonly available IBM or compatible personal computers. In addition to an overall description of *Facet* and its interactive modeling techniques, several underlying issues in the development of a modeling system are discussed. These issues include: the interactive user interface; 3-D display techniques; and ways of dealing effectively with model complexity. A number of areas for continued development, based on experience with the current implementation, are discussed.

KEYWORDS: configurable, interactive, three-dimensional, modeling, computer graphics, personal computer, design.

1. Introduction to *Facet*

Facet is an interactive 3-D modeling system for design professionals that runs on widely available personal computers (PCs). It allows a user to develop and manipulate 3-D computer models of shapes, spaces, and forms for use during the design process. Models are displayed on the screen in a *wire-frame* form while the user is interactively building or modifying the model. Selected views (including perspectives) may subsequently be rendered by the computer with shaded surfaces or may be output with a pen plotter or dot matrix printer. These two-dimensional pictures can then be the basis for working drawings or other enhanced renderings to be used for communicating the design to others.

Facet is initially directed to spatial designers who need to study alternative forms and shapes during the design process and create visual representations of physical systems. However, it will be readily adapted for use in other diverse areas of 3-D modeling such as generating input for graphic arts illustration systems as an aid in producing perspective drawings and renderings; building models, sets, and backgrounds for use by a 3-D animation system, when the elements being designed are static and do not require the expensive hardware necessary for real-time dynamic manipulation; and generating charts, technical

illustration, and other graphics to be incorporated with word processor output for document production.

A careful combination of features positions *Facet* as a readily available tool for 3-D modeling integral to the design process: a need that has not been adequately addressed affordably in the past. Some of the key aspects of *Facet* which recognize this need follow.

- Perhaps the most important aspect of *Facet* is the quality of interaction. The emphasis is on providing a modeling aid for use during the initial design stages. This is accommodated to a large degree in the way the designer is able to effectively interact with this electronic modeling medium. For example, the designer is able to quickly develop models of shapes, spaces, and forms; examine the model from many points of view to establish an accurate mental perception; and easily make changes in exploring different alternatives.
- A second key aspect of *Facet* is that it can deal with complex models. Unlike some previous modelers, project complexity is not limited by the computer's address space. Features are provided that enable the designer to organize the model with structure, grouping, and symbolic naming to work with abstractions and deal effectively with complex or detailed project models.
- Third, the ability to generate high quality shaded surface renderings, after interactively constructing the model, is an integral feature of *Facet*. The designer is able to assign various attributes to elements of the model that allow *Facet* to exercise the full capabilities of a high quality display device, if available, yet still operate effectively when using less capable display hardware.
- Finally, low cost is a significant feature of *Facet* to make it feasible for use by any interested professional designer. The basic personal computer system to support *Facet* can be purchased today for as little as \$2,000. (A much faster, full-featured system with the ability to generate full color renderings can be assembled for less than \$10,000.) Marketing strategy and pricing for the *Facet* software have not been finalized.

To be available to the largest group of users *Facet* runs under the PC-DOS (MS-DOS) operating system on IBM (or compatible) PCs. The minimum hardware configuration includes the basic PC with 640K bytes of main memory, a math coprocessor, a hard disk, a supported graphics display

subsystem and a locator (e.g. mouse). MS-DOS computers with faster or more powerful processors than the standard PC, but not as compatible, are also supported by *Facet* as long as an appropriate graphic display and locator are used. Any such increase in power will apply directly to improving the speed of interaction, for example, when calculating and displaying a new view of the model.

Facet supports several commonly used graphics devices including IBM's Color Graphics Adaptor and Enhanced Graphics Adaptor and the Microsoft mouse. Full color display systems (such as Number Nine's 512 X 32) are supported for smooth shaded color renderings. A variety of hardcopy devices are supported, including pen plotters, dot matrix printers, and film recorders.

2. Model Editing Techniques

Models in *Facet* are represented geometrically as polylines (one or more connected line segments) and polygonal surfaces. In addition, polygons are often joined together (sharing common edges and vertices) to form polyhedral shapes. Although curved lines and surfaces are not represented specifically, they are readily accommodated with the use of approximating polylines and polyhedra. (In particular, built-in primitive shapes are provided to quickly model arcs, domes and other common objects.) There is a display attribute that may be specified for such approximating polyhedra that causes them to be smoothed together (using Gouraud or Phong shading) when subsequently rendered with shaded surfaces.

Although the representation of models in *Facet* is fully three-dimensional, we are limited by our display hardware to visible windows on the model that are 2-D planes of projection. Likewise, we are limited by our pointing hardware to show locations in a single plane at any given time or to select or *pick* existing elements of the model by pointing to their displayed location on the screen. *Facet* employs two primary concepts in its operations for constructing and modifying models that minimize these inherent limitations:

- The first is to provide a *current plane* in 3-D space on which operations may be performed. For example, detailed surface features may be drawn on the face of an object after positioning the *current plane* as required. This *current plane* is readily positioned and oriented, as desired, and is displayed graphically on the screen with the model. For most operations in the *current plane* the user will find it convenient to select the orthographic projection with the *current plane* as the picture plane: making the surface of the display screen congruent with the *current plane*.
- The second concept is to do operations relative to the position or shape of existing elements in the model. For example, an object may be rotated to align with an edge of another existing object; the existing edge is readily *picked* by pointing at it on the display.

2.1. Current Plane Editing

A common strategy for constructing models with *Facet* is to begin by defining lines and polygons in space by working in selected positions of the *current plane*. Elements may then be connected or combined with each other by picking from the existing vertices to form additional lines or polygons. There is also a powerful *extrusion* operation

with which the user can sweep existing lines and polygons through space to form polyhedra whose cross sections are defined by the lines and polygons being extruded.

Work in the *current plane* is normally done graphically by pointing at desired locations on the plane with the locator (e.g. mouse). To provide an intuitive capacity for drawing on the *current plane* an orthographic view, with the *current plane* as the picture plane, is normally used so that the *current plane* is congruent with the display screen. The user can zoom the display in or out and pan around to focus on desired areas of the *current plane*. There are some unique aids to quickly and accurately select desired locations in the plane. These include:

- multiple grids of coordinates to which points may be automatically latched (with graphic definition of, and graphic feedback on, the operation of the grid's gravity field). By using two grids the effect of a rotated grid may be obtained.
- sets of slopes to which lines may be automatically snapped; and
- *background* graphics (derived from other views of the current model, or other *Facet* projects) with which points may be automatically aligned.

The coordinate locations generated by any of these automatic latching aids are, of course, calculated to the full resolution (32 bits) of the model's world space and are not affected by the resolution or zoom factor of the display being used. These aids may be grouped together as an *environment* to be saved under a user specified name for later use in the same plane or other planes or even in different *Facet* projects.

Most of the model change operations, such as moving, rotating, and changing size or shape, may be performed with either absolute locations or measures (e.g. selected graphically in the *current plane*) or with locations or measures relative to existing elements of the model.

2.2. Extrusion

A flavor of how *Facet* works may be gained by following the steps of an extrusion operation. Extrusion takes groups of polygons or edges and sweeps them through space to form new polygons. The sweep proceeds in a specified direction and is terminated by a specified plane.

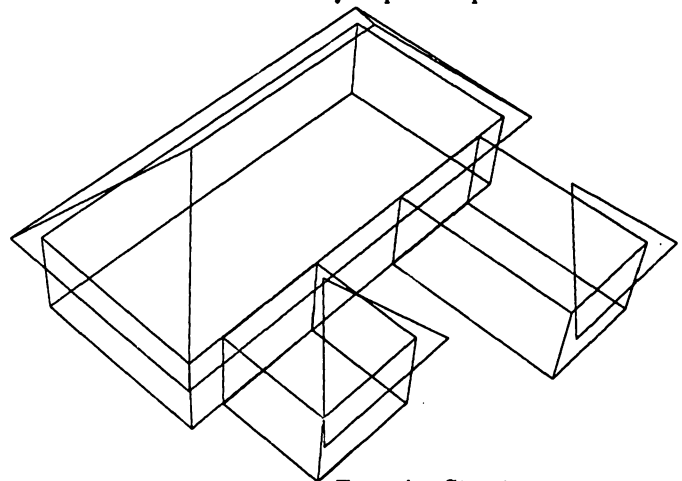


Figure 2-1: Extrusion Step 1

Figure 2-1 shows a simple building envelope being modeled in *Facet*; the pediments of the two projecting wings have been drawn by placing the current plane at an offset from the front of the wings. The user would like to extrude these triangles into the main roof to complete the form. First the direction of extrusion is specified. In this example the user picks an edge in the projecting wing to define the extrusion direction.¹ Second, the user defines the terminating plane. In the example the user picks three vertices that define the near slope of the existing roof over the main wing of the building.² Finally, the user selects the elements to be extruded. After the two pediments have been selected the extrusion takes place with results shown in figure 2-2.

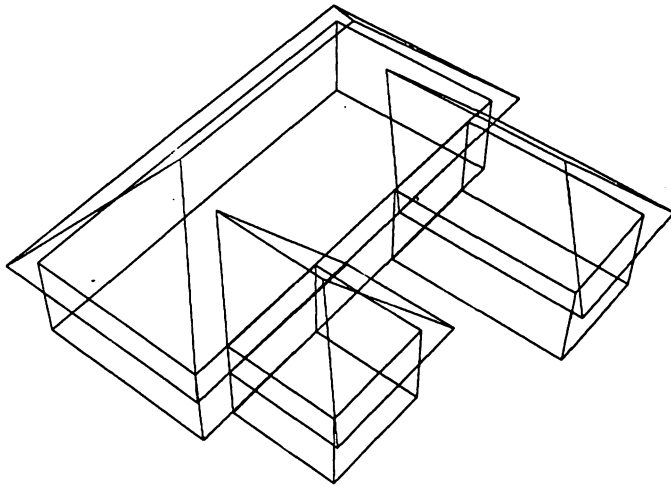


Figure 2-2: Extrusion Completion

The direction of an extrusion may also be specified by a polyline to create a multiple step extrusion. Here the terminating plane for the intermediate changes in direction at the polyline vertices is the average between the planes normal to the two adjacent edges of the polyline. An example of multiple extrusions is shown in figure 2-3 which shows a circle that has been extruded along the top of a building to form a pipe.

2.3. Programmable Commands

Incorporated in *Facet* is a command processor that permits a user to add new operations to customize the system. This goes beyond the macro capabilities of other systems to permit operations on the model representations that are not available as interactive commands. Most modeling systems such as *Facet* are organized to accommodate a large class of geometric shapes for parts, but the user is really working within a discipline that imposes restrictions on this class of shapes for a given type

¹In general, the direction of the extrusion may be the vector determined by picking an existing edge or two existing vertices or alternatively as the direction perpendicular to a plane or two edges (here it is really the cross product of the two edges).

²Likewise, the plane terminating an extrusion may be specified in one of several different ways: first, by using the current working plane; second, a plane that is parallel to but offset by a distance from the current plane; third, by picking three vertices in the model that lie in the desired plane; and fourth, the plane that passes through a selected vertex and is perpendicular to a selected edge.

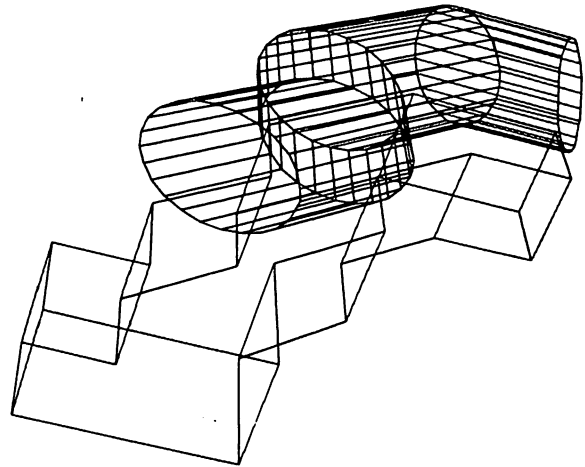


Figure 2-3: Extrusion along a Polyline

of part. Knowledge of such restrictions may be used to advantage in the definition of operations provided for the particular type of part. Operations restricted in this manner make it easier to construct valid examples of such parts. For example if a particular rectangular solid represents a wall, the different compositions that may be constructed by the user using that rectangular solid should be restricted to those that are possible with walls. Restriction of the possibilities that may be modeled during the design of an artifact makes the modeling task faster [4]. In order for a modeling system to speed the task of the user it should take advantage of any inherent restrictions and define the manipulation operations so they reflect those that may occur on the actual artifact.

3. Flexibility in User Interface

Perhaps the most important aspect of *Facet* is the quality of interaction. The emphasis is on providing a modeling aid to the designer for use during the initial design stages. This is accommodated to a large degree in the way the designer is able to effectively interact with this electronic modeling medium. For example, the designer is able to quickly develop models of shapes, spaces, and forms; examine the model from many points of view to establish an accurate mental perception; and easily make changes in exploring many alternatives.

The user's interface to *Facet's* operations is at the same time easy to learn and effective in use for the expert. All operations may be accessed through mouse activated menus (which may be restructured by the designer to fit any particular desires). In addition, operations may be bound to any of the keys of the keyboard for invocation at the push of a button. Any operation may also be accessed by explicitly typing in a unique abbreviation of the operation's name or selecting it from a scrollable window containing descriptions of all the operations.

The style of menuing used is *pull-down* or *drop-down*, similar to those used in several recent integrated operating environments (such as Apple's Macintosh [7]). The operations are grouped into several small menus according to some similarity of use and each menu has a name. Small groups of these menus are clustered together to be accessible to the user simultaneously. Each operation is represented in the menus by a descriptive phrase (i.e. menu button). Normally, the display screen is filled by the

window displaying the 3-D model except for a single line of text at the top containing the names of the menus available in the currently active cluster. By pointing to a menu name, the user *pulls down* the menu whereupon the list of operations available in that menu appear immediately on the screen. The user drags to the desired operation to select it and, on doing so, the model display window is immediately restored and the selected operation begins. Within the individual operations, additional input requested of the user is performed graphically, whenever appropriate, by pointing to locations in the *current plane*, by *picking* existing elements of the model, or by answering a question from a multiple choice *dialogue menu*. In addition to activating a particular operation, selecting an item from a menu can also be used to effect a change to a different active menu cluster or a change in the contents of a pull-down menu.

There has been a strong commitment, beginning with the initial designs for *Facet*, to a configurable user interface. This is primarily embodied in the ability of the users to define their own set of command menus. In the first place, the users may define the names used in the menus to represent the commands. Second, they may define the grouping of commands to form menus and the names of the menus. Third, they may define the clustering of the menus. Finally, the dynamics of when the clusters and menus change may be fully customized. Configurable interaction is provided in order that the designers may tailor the interface according to their specific needs and desires since each designer might evolve an uncommon approach to design using this modeling medium. In addition, an individual designer will find it useful to organize the interaction differently for varying types of projects.

Brown [1] has defined a methodology that might be used to specify the structure of a menu network. His approach is not different from the one used in *Facet*. Both approaches define the structure using a predefined syntax but *Facet* does not provide any mechanism for explicitly controlling the complexity of the menu structure. We did not wish to impose, at the time of implementing the menu package, any untested assumptions about what the desirable properties of a menu structure would be for our applications. Because of this uncertainty about what constitutes "good" structuring we provide one low-level structuring tool that in Brown's terms is the GOTO statement.

If the user is going to be permitted to repartition his menus and order them to his liking there must be some discipline imposed on the structure of the application program. This may be done through the syntax of the menu structure definition. It is well known that these problems of structure definition exist and many people have proposed solutions [6, 8]. *Facet* is implemented using yet another solution which will be described in more detail in a future paper.

4. Model Display Techniques

As a physical system modeling tool (which might often be used during the design process) it is important for *Facet* to provide the user with a strong perception of the shapes and forms being modeled. This visual perception is readily gained through the ability to look at the 3-D model from any desired point of view. *Facet* displays selected parts of the model (the *working set*, as presented in the following

section) on the screen as a *wire frame* (i.e. with the edges of each polygon drawn as lines) while the user is interactively constructing or modifying the model. The user is able to select a view graphically by pointing to the desired location of both the viewing position and the center of interest. Alternately, orthographic views may be selected with the picture plane placed in the *current plane* (which may be readily positioned to any desired location in 3-D space).

Selected views may be saved for later use in *view files*. Although the specifications stored in such files represent a 2-D projection of the 3-D model, they are not specific to the particular display screen in use when they were saved in order that the views may be subsequently displayed on other devices including pen plotters and dot matrix printers. In addition to the location of the elements in the 2-D view, the *view files* also contain depth information so they can be displayed with polygons rendered as shaded surfaces with any hidden surfaces removed. Also, *view files* can be automatically regenerated, on request, to update a previously saved view after the model has been modified.

5. Effectively Dealing with Model Complexity

Facet is able to accommodate large complex models. However, complexity by itself is of no advantage: it degrades the speed of interacting with the model and leads to visual clutter, logical confusion, and chaos. In order for the user to deal effectively with complex models, *Facet* provides several mechanisms for structuring and organizing the elements of a modeling project. *Facet* allows the user to develop abstractions and aggregations to work with selected aspects of the model in isolation from other aspects. In *Facet* there are several basic mechanisms for organizing the model: *the family tree*, *collections*, *symbolic naming*, and *the working set*. Some CAD systems, usually those based on automated drafting and used primarily to generate 2-D drawings, provide a simple method of partitioning a project (i.e. drawing) into several separate entities (usually called *layers* or *overlays*), a technique that is derived from traditional drafting methods. In order for a 3-D modeling tool, such as *Facet*, to be used effectively with complex projects, richer ways of organizing the modeling medium are required.

Complex models are at the same time a necessity and a burden. Models of complex artifacts are made of many discrete components that often have overlapping conflicting design criteria. It is often desirable to accurately model many of these components and features. On the other hand, we want to limit the complexity to something that is manageable and still useful as an aid to analysis and communication; time and cost will also put a limit on the desired complexity.

5.1. Abstractions and Aggregation

A useful rule in balancing between the opposing needs for complexity and simplicity is to only model enough detail to adequately predict the *performance* of the artifact³ and to adequately communicate the model to others. Even with a conscious effort to limit model complexity the level that the human user is able to deal with effectively is quickly

³The performance required of an artifact might be to satisfy some form of aesthetic evaluation as well as structural, functional, or other types of more readily calculated performance.

exceeded. We must accommodate this overload by learning to make abstractions and aggregations.

All models that fall short of embodying the emulated artifact itself are in one way or another abstractions. When abstractions eliminate the need to deal with details that are irrelevant at the moment, the user is free to concentrate on a few related criteria at a time. For example, two simple boxes of appropriate proportions might be used as an abstraction of the World Trade Center to do site positioning studies. Often, it is helpful to abstract an abstraction and so on, establishing many layers or a hierarchy of abstraction.

With aggregation we can group items similar in feature, purpose, or design, allowing us to work with the group efficiently, in isolation from other items and groups, and apply operations to the entire group in a like fashion. For example, advantageous groupings in a building might include the structural, electrical, or mechanical components; spaces serving a similar function, such as stairways; or different components from the same manufacturer.

5.2. Model Organization in *Facet*

The previous requirements drove the development of the four organizational mechanisms provided by *Facet*. These are the *family tree*, *collections*, *symbolic naming*, and the *working set*.

The *family tree* allows a hierarchical organization of the elements of a model. The nodes of the tree are called *members* with each having a *parent* member, a set of zero or more *sibling* members (all having the same parent), and a set of zero or more *child* members. At the base of the family tree is the *root* member that does not have a parent, is an ancestor of all other members in the family tree, and is provided automatically in a new project. The family tree is inherently a spatial organizational mechanism. In the first place, all parts, or spatial elements, of the model exist within the hierarchy of the family tree. In addition, the content of each member in the family tree (other than the hierarchical structuring information) is a location, orientation, and scale in 3-D space that is applied to an optionally referenced part description (called a *prototype* in *Facet*). Each prototype may be referred to by one or more members allowing many discrete instances of similar parts with only a single description being stored. Finally, the location/orientation/scale of a particular member is not defined in *absolute* terms within the 3-D world, but is *relative* to that of its parent in the family tree. Thus, an operation performed on a particular member may also affect all descendants of that member. For example, moving a member to a new location may be done in a fashion that effectively moves all descendants in a like way so that the relative positioning of the member and its descendants remains the same.

A second mechanism for grouping members, without any implied spatial connectivity, is provided with *collections*. Collections offer a means of assembling aggregations of members for any desired purpose in a manner orthogonal to their structure within the family tree. A member may be included within several different collections or, on the other hand, is not required to be in any collection. Other collections may be included in a collection, allowing hierarchies of collections.

The third mechanism, *symbolic naming*, allows the user to associate a name (any string of up to 60 characters) with a particular member or collection. Selection of such a component may be done by specifying its name. *Facet* will also, in certain situations, identify components to the user by displaying their names.

The last organizational mechanism is a dynamic sub-set of the project model, called the *working set*. This is the set of components that the user has selected to work with together at a particular point in a modeling session. All the components in the working set are displayed on the screen and are readily accessible by the user through the *Facet* operations. By the same token, components not in the working set are not displayed and are not generally accessible without placing them in the working set. There is a set of operations in *Facet*, collectively called the *project browser*, that allows the user to scan through the entire project model and move components into and out of the working set. The project browser permits the user to search through any accessible project and copy parts of other projects into the current one.

5.3. Modeling Medium Limitations

In addition to addressing the conceptual difficulties of effectively working with complex models, we must also deal with the physical limitations of our modeling medium. For systems with an electronic computer performing an essential role, such as *Facet*, the principal limitations are the memory size and processing speed. The size of memory regulates the amount of model information that may be displayed and manipulated simultaneously. The processing speed affects the smoothness of interaction, e.g. when redisplaying the model from another viewpoint, selecting an item, or performing a spatial modification. The amount of processing required for such operations is directly proportional to the complexity of the model on display.

Some modeling systems choose the route of limiting a project's size so that it will fit into the computer's main memory (examples of this are Design Board Professional by Mega CADD Inc. [3] and Polycad/10 by Cubicomp Corp. [2]). These might be called part modelers as opposed to assembly modelers. Others use a simple form of virtual or paged memory, pretending that the main memory is larger than it really is and storing it instead on the slower disk memory (examples of this include Microcad by Imagimedia Technologies Inc. [5]). As in any system that saves project models from one session to the next, *Facet* stores its models on disk. *Facet* does not limit the project by the size of main memory. It uses the organizational mechanisms described above to allow the user to select parts of the model to be operated on together at any given point (perhaps a set of abstractions or aggregations). This *working set* is then resident in main memory and does not incur the slowdown of continually reading and writing the data on disk as in a demand paged memory scheme, that is not able to take advantage of the user's logical organization of the project data and working patterns. The user will normally keep the working set as small as possible, including only those components relevant to or providing context for the current operations, to minimize the processing time for interactive operations. *Facet* includes a *project browser* with that the user may alter the working set (by adding or removing items) at any time during a modeling session.

6. Future Directions for Facet

Although *Facet* is now a complete system, the area of 3-D modeling is very wide ranging and we have several additional enhancements in progress and planned for development over the next several years. In the area of interactive model construction and editing techniques developments that are planned include:

- Suites of application specific modeling operations and primitive objects (ultimately user defined primitive objects provided through application of a programming facility).
- Additional general purpose modeling operations, e.g. patterned replication, polygonal meshes, spatial reconstruction (digitizing 3-D shapes from multiple 2-D views).
- Use of image digitization from hardcopy, film, video tape, or video camera, for use as surface texture on 3-D model elements or as backdrops for the model where the synthesized perspective projection of the project model is matched to that of the scanned in scene.
- Interactive rendering enhancement with *paint system* type of techniques and 2-D images as backdrops.
- Solid modeling - using a winged-edge type of polyhedral boundary representation currently running under Unix (with robust spatial set operations).

In the area of general interactive control and menuing, we will be experimenting with ways of providing a displayable map of the entire menu structure, with the active menu cluster highlighted, to provide an effective aid to navigating a complex network.

The next step would be to provide an interactive menu structure editor that operated directly on the displayed graphic representation.

Continuing developments are planned in dealing with the graphic display and access of *Facet* model data. Several enhancements to the shaded surface rendering are under way. To transfer graphic data to drafting systems, paint systems, and other picture enhancement programs, output will be provided in IGES and widely used proprietary formats. (Currently a library of routines is provided for use by client programs that allow direct access to the *Facet Disk Data Structure*.) The current simple form of interactive animation set up provided to interpolate camera views through key poses will be enhanced to allow more flexibility of movement. With the use of new high performance hardware display processors (both on the PC and other hosts for *Facet*) the playback of these animated sequences will be speeded up to near real-time; interactive operations will also benefit from increased dynamic motion and speed of redisplay.

7. Conclusions

In *Facet* we attempted to provide a modeling tool that is accessible to many designers during the conceptualization of a project. We have done this by presenting operators that make generation and change of models fast and simple. Use of the system has shown that it has the depth and flexibility to easily adapt, extending the habits and design processes of an individual designer.

References

- [1] Brown, J.W.
Controlling the Complexity of Menu Networks.
Communication of the ACM, July, 1982.
- [2] *Polycad/10 User's Manual - Version 2.0*
Cubcomp Corporation, Berkeley, Ca, 1985.
- [3] Engelke, D.J., Heitzman, F.E., Voosen, J.C.
3-D Program Reviews.
Architectural Technology :39-42, January/February, 1986.
- [4] Glass, G. J.
Automated Part Location in the Design of Assemblies.
Technical Report CSL-83-4, Institute of Building Science, Carnegie-Mellon Univ., January, 1983.
- [5] Hart, G.
Complex CAD: Software for the PC.
PC Magazine 5(5):111-148, March, 1986.
- [6] Kasik, D.J.
A User Interface Management System.
Computer Graphics 16(3), July, 1982.
- [7] *Macintosh*
Apple Computer, Inc., 20525 Mariani Ave., Cupertino CA 95014, 1984.
- [8] Price, L.A.
Design of Command Menus for CAD Systems.
In *Proceedings of the 19th Design Automation Conference*. Las Vegas, June, 1982.

Psychology and the User Interface:

Science is soft at the frontier

(Abstract of invited talk)

John M. Carroll
User Interface Institute
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598, USA

One source of intellectual overhead that every science inflicts on itself periodically is the clarion call to "be hard", to establish methodological ground rules so severe that they will insure that good science can prevail. This romantic notion would only be that if it were not for the fact that these fits of methodological purification have typically led to conceptual and empirical poverty. The excesses of positivism and its crippling effects on the sciences from physics to psychology are still in recent memory.

Newell and Card, in an invited article in the journal *Human-Computer Interaction*, have undertaken a modern variant of this methodological cleansing. However, in most respects their motivation and arguments are precisely those of the positivists. They urge that the psychology of human-computer interaction needs to be hardened, meaning it must more uniformly subscribe to parameter fitting, calculation, and quantitative approximation. They are explicit in identifying as their motivation the fear that the "harder" disciplines of user interface design and artificial intelligence will not take usability psychologists seriously unless the psychologists have hard methods. They suggest a modified Gresham's Law: "hard science drives out the soft," as if this is both inevitable and a good thing.

My own view is that science is always soft at the frontier. The psychology of human-computer interaction is at a frontier of method and theory in psychology and a frontier of technology and application in computer science. To me, it is fantastic to insist that we start right out on a "hard" psychological theory to guide designs for integrated co-authoring applications on workstations that support multimedia input/output when we can barely couch such a theory for well-worked, toy domains like cryptarithmic and chess.

Newell and Card are too concerned with the form of science and too little concerned with its content. They urge calculation and quantitative approximation but seem almost blase about what exactly is calculated or approximated. At best, Newell and Card's discussion is very premature; more likely, it threatens to set the psychology of human-computer interaction backward by confusing the project of developing

a fundamental understanding of usability and user psychology with the engineering practices we might be able to develop if we had such a science base to begin with.

This talk has four parts. In the first, I consider Newell and Card's clarion call for hard science, reviewing a critique developed jointly with Robert Campbell of IBM Research. Campbell and I argue: (1) that Newell and Card misunderstand and underestimate how psychology currently contributes to interface design and thus set out to solve a nonexistent problem; (2) that they misunderstand and oversimplify the system design process, and that indeed only by doing so can they find a role in it for their clumsy hard science; (3) that their replies to existent criticisms of their hard science are uniformly without serious content.

Their reply to the charge that their hard science is too low level is essentially to redefine "psychology" so that it perfectly coextends with their enterprise, leaving critics to attack psychology and not them. Their reply to the charge that their hard science is too limited in scope is to try to assimilate a variety of current work (much of it not so low level) to their enterprise merely by saying "it fills out our 'vision'." (Notably, these two replies, taken in conjunction, are self-contradictory). Finally, their reply to the charge that hard science takes too long to help at all in the development process is to say that the elaboration of interface technology in fact takes place more slowly than everyone thinks it does!

In the second part of the talk I examine some of the current research work in human-computer interaction that is paradigmatically hard. I argue that the psychology of human-computer interaction, like psychology generally, suffers from a methodological bias for posing elegant, either-or research questions that idealize away variables like task context, e.g., "is mouse driven pointing control better than a velocity control joystick?" Perhaps the question should be: "under what circumstance is a mouse the right design choice, and under what circumstance is a velocity control joystick the right choice?" Hard psychologists seem too willing to trade off ecological scale for laboratory tractability (e.g., a study of command languages that exam-

ines a command set of 3 commands when realistic scale would be 1-2 orders of magnitude larger). The hard science of Newell and Card rests fundamentally on baldly unreasonable idealizations (e.g., assuming errorless performance for purposes of theory when in fact obtained error rates exceed 30 percent).

We need to concentrate on the important facts of user behavior, not ignore them because they lie outside our methodological purview. We must of course strive to make our science harder (in the usual sense of "more systematic"). But we must also guard against too much weight being given to superficial rigor and too little to the practical value of our theories in guiding the design of new technology.

In the third part of the talk, I examine the area of artificial intelligence research specifically directed at the construction of advisory expert systems (intelligent help and training facilities). Newell and Card might find it startling that an domain in the mainstream of AI, which they describe as hard, in fact has no systematic theoretical foundations (no constitutive theories of types of general skill, no principled taxonomy of knowledge domains, no user models that do not obviously violate fundamental facts about human learning and the growth of knowledge).

Indeed, this supposedly hard research area has no comprehensive methodology: experimental systems are routinely designed with the paramount goal of providing advice to users without any systematic consideration of how people give and take advice, what their real problems, goals, or needs are, etc. The field has a large inventory of dialog techniques, for example, but no understanding of the circumstances under which particular techniques are useful or of how to integrate various techniques to capitalize fully on prior work. Finally, there is no effective engineering aspect to this work: no one knows how to develop advisory expert systems with limited resources or on short schedules. It is simply incredible that anyone who understood the state of art in this field could hold it up as a paradigm of hard science.

In the final part of the talk, I consider what human-computer interaction might need in the way of a soft science, a conceptually richer and methodologically less limited science. I urge that we recognize that in a rapidly evolving, technology-driven area hard science can never drive out the soft. Rather it consolidates those areas that have become well-worked. We must learn better how to use soft science

to identify concepts and behavioral phenomena that are really worthy of quantification and other "hard" analysis. We must develop an arsenal of realistic empirical methods, methods that efficiently and reliably produce information at the right level to impact the application of new technology, not merely at a convenient level. For example, collecting and taxonomizing users' critical incidents or thinking aloud protocols may generate information more directly pertinent to an iterative design process than a record of individual keystroke times -- but the keystroke times are "hard," more convenient to collect, and more familiar and routine to analyze.

We must develop qualitative theories, and means for expressing such theories. For example, a list of user knowledge states, with associated transition rules, may be more relevant to guiding the design of new technology than an equation describing a fitted curve of millisecond differences between performance means. Finally, we must extend the scope of our theories. For example, if users routinely make many errors, then our theories should incorporate errorful as well as errorless behavior. Empirical taxonomies of error, and even rough theories of action slips, abductive reasoning, and learning via metaphor and analogy are soft science, but perhaps critical if we are to have a serious and effective science of human-computer interaction.

In summary, it is elementary in the history of science that one cannot legislate the quality of the conceptual and empirical *content* of science merely by legislating the methodological *form*. In fact, if history is any gauge, *a priori* limitations on acceptable methods usually have an undermining effect on conceptual and empirical quality. Newell and Card are mistaken in their attempt to confine the psychology of human-computer interaction. Their view of hard science is arbitrary and in particular has been a fairly well-documented failure in providing real leverage in interface design, conceptually and empirically. Their view of AI as hard is similarly inaccurate, as evidenced by the subfield of advisory expert systems. Finally, there are routine alternatives to their hysterical and dismal clarion call.

Gresham's Law states that "bad money drives out the good", but it does not suggest that we accept this as our inescapable fate. Rather, it suggests that we protect good money by responsible fiscal policies. I suggest that we protect soft science by responsible methodological policies. Whenever a scientific program is championed on purely methodological grounds, we should cringe.

LEARNING GRAPHICS PROGRAMMING BY DIRECT COMMUNICATION

Martin Tuori
Tim Pointing
Defence and Civil Institute of Environmental Medicine
PO Box 2000
Downsview, Ontario, M3M 3B9

ABSTRACT

The process of learning the graphics functions of a computer graphics workstation environment is both assisted and hampered by the presence of an intermediary programming language. Assistance comes in the form of programming language functions for preprocessing, storage declaration, expression evaluation, control flow, and system libraries. Working against the student, compilation of programmed examples is slow, and errors may arise both from the syntax and semantics of the graphics functions, and from those of the programming language.

We propose an approach to learning the graphics functions that temporarily separates the graphics component from other aspects of the overall programming environment; in a sense, we are proposing training wheels for the graphics subsystem.

This approach was used in creating, for the IRIS series of workstations,¹ a graphics interpreter that allows a student to test out graphics concepts, without the need to write programs. Subroutine calls typed to the interpreter are carried out immediately, allowing a quick, trial-and-error approach. We argue that this approach is a useful addition to conventional learning techniques, and that its success can be attributed to bringing the student programmer into more direct communication with the graphical components of the programming environment.

INTRODUCTION

A student learning the details of a new computer graphics programming environment may employ many different techniques. He may begin by reading the manufacturer's documentation, which, if well written, conveys basic concepts, syntax, semantics and suggestions for efficient use of the computer graphics system. While this is an important stage in the student's training, it is not enough to give him fluency as a graphics programmer. Writing small test programs is good way to proceed, and is made much easier if a sample skeleton program, or stub, is provided. As Duff says, "Whenever possible, steal code." [Duff 1985]. The student can extend the stub to exercise individual features of the graphics environment, or combinations of features, thereby gaining familiarity with the concepts and behaviour of the system.

A high-level programming language provides a variety of features that can help the student in his exploration of a system and its functions. Macro preprocessing serves two useful functions — common constants and expressions are provided in system files, for inclusion in new programs, and the student can define macros to suit his own needs. Storage declaration provides for complex object definition, and for loading them from external sources. Expression evaluation allows results from one operation to be used as input to another; for example, reading pixel values from a raster display, modifying and redisplaying them. Features for control flow allow conditional, iterative and recursive action. Finally, various support libraries for mathematical, input/output, networking and other functions offer specific functionality, as needed. Although the student can defer the use of some language features, such as specialized subroutine libraries, he cannot avoid the basic syntax and semantics of the programming language itself.

1. IRIS is a trademark of Silicon Graphics Inc.

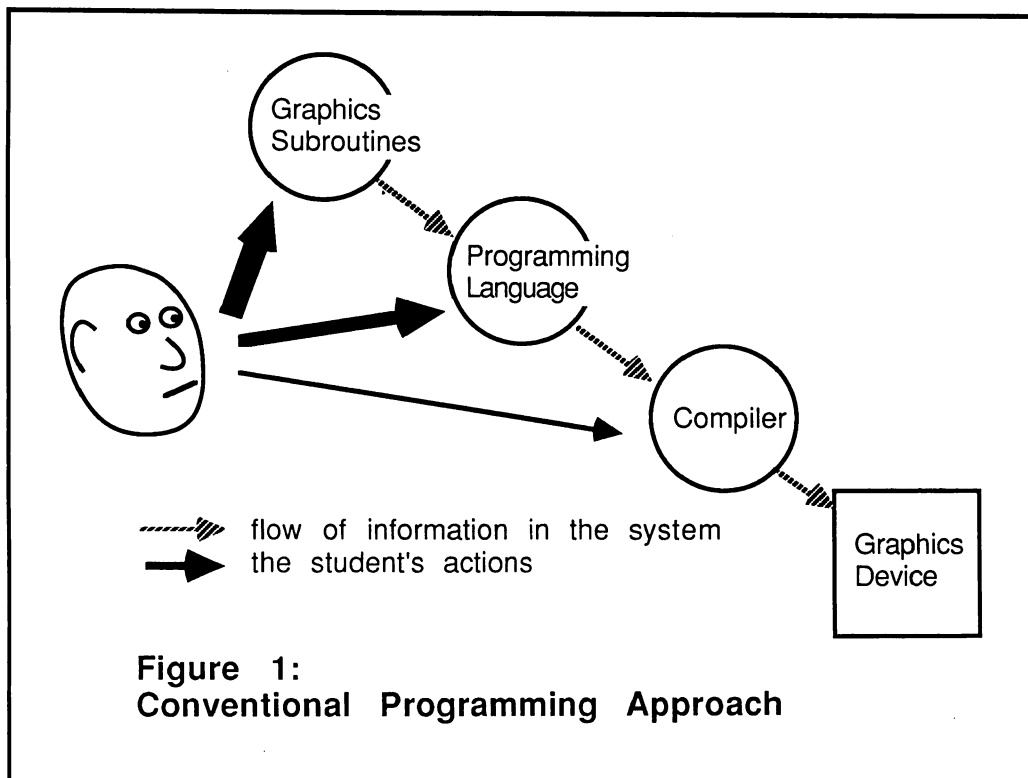
Many of the basic language features are of little interest, initially, to the student studying a graphics subsystem; rather, he needs to concentrate on the graphics subroutine calls. The *programming approach* is error-prone, tedious and time-consuming. The student's efforts at writing even small, correct programs are invariably delayed by errors in the syntax or semantics of the programming language; these must be corrected by repetitive editing, compilation and testing.

These problems arise because the programming approach is *indirect*. The student needs to test his skills in using the graphics functions, but is forced to do so through an intermediary, albeit high-level, programming language (Figure 1). If the programming language is interpretive (some implementations of Basic, Lisp, APL, etc.), test runs can proceed quickly; in many cases, however, the programming language is compiled (most implementations of C, Pascal, Fortran, etc.), and considerable time is spent waiting for compilation to take place. Since the student's efforts are highly exploratory, characterized by tens or hundreds of trial and error steps, considerable time and system resources may be wasted.

A DIRECT INTERPRETIVE APPROACH

Recent literature on Human-Computer Interaction (HCI) has promoted the use of *direct manipulation* [Shneiderman 1983], [Kay 1984], [Hutchins, Hollan and Norman 1986], [Witten and Greenberg 1985]. Shneiderman characterizes direct manipulation by: the visibility of the object of interest; rapid, reversible, incremental actions; and replacement of complex command language syntax by direct manipulation of the object of interest.

The situation here is different, in that there is no easily defined *visible object of interest*. The student is studying the process, or language of graphics programming. Perhaps the conventional approach, in which we use a complex command language (the programming language interface) should ultimately be replaced by more more visual, manipulative, or demonstrative programming methods. We are somewhat constrained, however, by the present state of programming support on graphics workstations; the student must learn to control a graphics system through a highly linguistic interface. Our objective here is not to introduce direct manipulation, but to offer *direct communication* between the programmer and the graphics library.

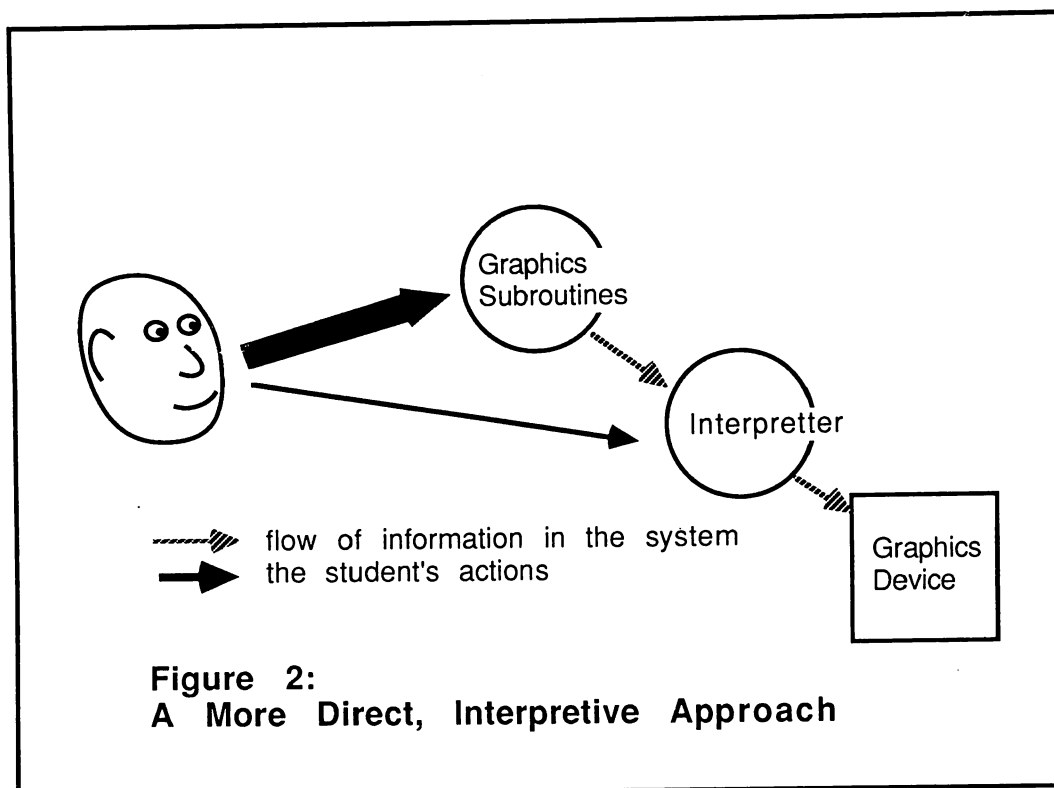


A student's initial, exploratory efforts are better supported by a fast, interpretive interface (Figure 2). The student should be able to compose requests for graphic subroutine calls, and have them carried out directly, with immediate visual results. The approach is not new, but can be dated back at least to Turtle Geometry [Byte 1982], [Papert 1980]. The work presented here is not intended to teach children to use computer graphics, or to teach them problem-solving skills; it is intended to teach the programming details of two- and three-dimensional shaded graphics in environments like the IRIS workstation [Silicon Graphics Inc. 1984].

A graphics interpreter should act as an additional tool in the student's kit. As he progresses, he will need to try writing real programs; this transition is easier if the language of the interpreter corresponds, as closely as possible, with the style of programming that will ultimately be demanded of the student. Although there is a temptation to provide additional functionality in the form of high-level primitives for drawing, menu-driven interfaces, etc., this must be resisted, unless those primitives are part of the toolkit the student will later use. The objective here is not to create yet another language for graphical expression, but to mimic, as closely as possible, the existing graphical component of the high-level programming language.

We have constructed an interpreter, for the IRIS workstation, called *irisinterp*, or *ii*. In *ii* the following sequence of commands produces a perspective view of a coloured box with a white top:

```
perspective(600,1,1,2000)
makeobj(1)
/* a tall red box */
color(1)
polrf(5, 0, 0, 0, 10, 0, 0, 10, 0, 40,
        0, 0, 40, 0, 0, 0)
polrf(5, 10, 0, 0, 10, 10, 0, 10, 10, 40,
        10, 0, 40, 10, 0, 0)
polrf(5, 10, 10, 0, 0, 10, 0, 0, 10, 40,
        10, 10, 40, 10, 10, 0)
polrf(5, 0, 10, 0, 0, 0, 0, 0, 0, 40,
        0, 10, 40, 0, 10, 0)
color(7)
polrf(5, 0, 0, 40, 10, 0, 40, 10, 10, 40,
        0, 10, 40, 0, 0, 40)
closeobj
color(0)
clear
lookat(45,45,50,0,0,15,1150)
callobj(1)
```



Punctuation, and other syntactic details are relaxed in *ii*; trailing semicolons (signifying the end of a statement in C), parentheses for subroutine arguments, and commas are treated as white space. Readers familiar with the IRIS programming environment will recognize that, with a few changes in punctuation, this sequence could be turned into a C program to carry out the same function. In fact, it is part of a longer sequence to draw the coloured boxes example provided in the manufacturer's documentation. With this sequence, the student can more easily follow the documented description of three-dimensional viewing controls, use of the z-buffer, etc.

Our early experience with *ii* led us to extend the basic concept, somewhat, to include the following features. A script inclusion feature has been added, by which a file containing *ii* instructions can be called, for insertion into a sequence; for example, the following sequence performs simple animation by calling the *boxes* script, and then rotating it by 5 degrees about the z-axis:

```
script(boxes)
color(0)
clear
rotate(50z)
callobj(1)
color(0)
clear
rotate(50z)
callobj(1)
color(0)
clear
rotate(50z)
callobj(1)
...
```

This allows longer sequences to be prepared with a text editor, tested and refined. It also allows the development of a set of tutorial examples. Scripts may be nested to a pre-determined limit; but recursion is ineffective, due to the lack of a method for expressing conditional termination. Standard defined constants are provided, for boolean values, colours, and screen limits:

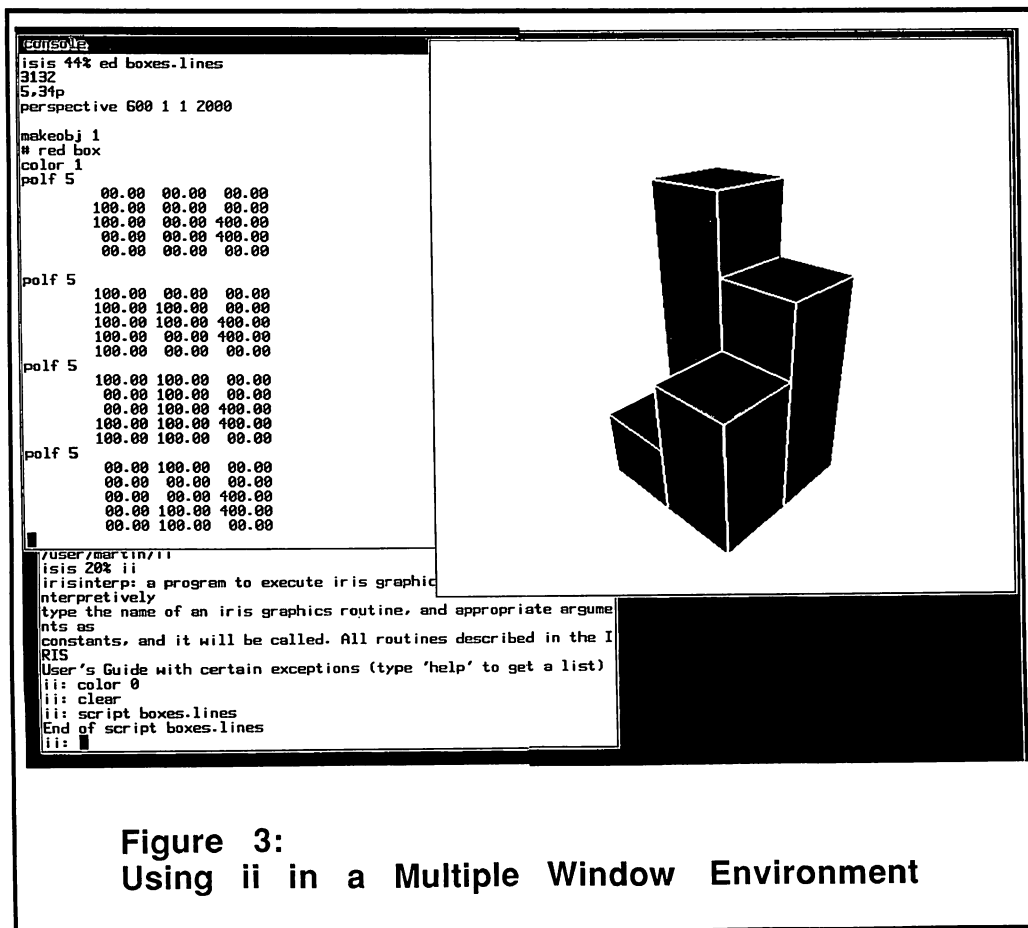


Figure 3:
Using *ii* in a Multiple Window Environment


```
color(magenta)
move(0,0,0)
draw(xmaxscreen,ymaxscreen,0)
```

We have resisted the temptation to add declarative and iterative capabilities, partly because of the implementation effort they would require, and because the student who is ready to use those features is ready to move on to programming in C.

A few subroutine calls from the IRIS GL-2 graphics library are not supported in *ii*, because they are inappropriate in this context. For example, the subroutine *callfunc()* requires the address of a C subroutine, to be called from within a graphical object; the student using *ii* has no way of determining such an address. Similarly, the subroutine *defrasterfont()* requires an array containing the bitmap definition of a raster font; the student cannot be expected to type in such an extensive data structure.

The *ii* program has been useful in our efforts to explore and understand the IRIS programming environment. For example, details of the window-to-viewport coordinate transformation were initially confusing; trial and error with *ii* helped considerably. Interactions between z-buffer and double-buffered display techniques were also explored easily using the interpreter. As our skill at programming the IRIS increases, we still return to *ii* occasionally to check out details of some graphics functions. In the multiple window environment (MEX) of the IRIS, it is easy to digress from a programming task to try out an idea using *ii*, and then return with the solution in hand. An example is shown in Figure 3, in which a text editor (upper left) is being used on the boxes script, the *ii* program is being run from a partially obscured window in the bottom left, and the graphical output from *ii* is at the upper right.

The development of *ii* was, not surprisingly, a tedious task. Data typing in C is sufficiently strong that subroutine calls, with arbitrary numbers and types of arguments, cannot be assembled and executed dynamically. It was necessary to group subroutines by their calling-sequence patterns, and use a common stub to assemble appropriate arguments and dispatch the request, as appropriate. For example, routines that take four short integers as input form one group, while those that take four pointers to short integers form another. In all, the source code for *ii* takes 30 pages; the compiled program is quite large, at 164 k-bytes, since it includes most of the GL-2 graphics library.

The savings afforded by *ii* are significant. A short sample program, that draws a three-dimensional cube intersected by a plane, occupies 1,390 characters of text in *ii*, whereas the C source takes 2,119 characters. Compilation in C takes 37 seconds on an IRIS-2400 (no other users), and the compiled program is 61,440 bytes long.

CONCLUSION

In this paper, we have described a learning situation (graphics programming) in which the student's efforts are hampered by the insertion of an intermediary, high-level programming language and its support environment. A direct, interpretive approach improves the speed of learning, by bringing the student into closer contact, or communication, with his objective — the syntax and semantics of the graphics subroutine library he is trying to learn. *Direct communication* is an adaptation of the concept of *direct manipulation*, for situations where the user's objective is not a visible entity, but a linguistic process.

References

- Byte, *Special Language Issue on LOGO*, McGraw-Hill, Aug 1982.
- Duff, T., Quoted in *Programming Pearls* (Jon Bentley), *Comm. ACM* 28, 9, (Sep 1985), 896-901.
- Hutchins, E. L., Hollan, J. D. and Norman, D. A., *Direct Manipulation Interfaces* (to be published), in *User Centered Systems Design: New Perspectives in Human-Computer Interaction*, D. A. Norman and S. W. Draper (Eds.), 1986.
- Kay, A., Computer Software, *Scientific American* 251, 3, (Sep 1984), 52-59.
- Papert, S., *Mindstorms: Children, Computers & Powerful Ideas*, Basic Books, New York, 1980.
- Shneiderman, B., *Direct Manipulation: A Step Beyond Programming Languages*, *Computer* 16, 8, (Aug 1983), 57-69.
- Silicon Graphics Inc., *IRIS User's Guide, Version 2.0*, Silicon Graphics Inc., Mountain View, CA, 1984.
- Witten, I. H. and Greenberg, S., *User Interfaces for Office Systems*, Research Report No. 84/161/19, Man-Machine Systems Laboratory, Dept. of Computer Science, The University of Calgary, Feb 1985.

VLSI AND GRAPHICS AT THE PIXEL LEVEL*

Henry Fuchs

Department of Computer Science
University of North Carolina
Chapel Hill, North Carolina 27514

Abstract

The computational bottlenecks in many interactive raster graphics systems are the pixel-level calculations and not the display list traversals, geometric transformations or clipping computations. We examine several VLSI-based designs that focus on these pixel-level calculations, noting the influence of the price-performance target on system design and component selection. We describe in detail the latest results from *Pixel-planes*, our experimental system optimized for algorithms in which many of the pixel-level calculations can be formulated as linear expressions ($Ax + By + C$) of the pixel's x,y address. We outline variations on the current implementation for greater generality or faster speed or lower cost. We show the effects of these changes on the algorithms that are to be run on the alternative implementations.

Bibliography

Clark, J.H. and M.R. Hannah, "Distributed Processing in a High-Performance Smart Image Memory," *Lambda* (now *VLSI Design*), 4th Quarter, 1980.

Fuchs, Henry, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Frederick P. Brooks, Jr., John G. Eyles, and John Poulton, "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-planes," *Computer Graphics*, Vol. 19, No. 3, July 1985 (Proc. SIGGRAPH 85).

Goldfeather, Jack and Henry Fuchs, "Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory," *IEEE Computer Graphics and Applications*, Vol. 6, No. 1, January 1986.

Goldfeather, Jack, Jeff P.M. Hultquist, and Henry Fuchs, "Fast Constructive Solid Geometry Display in the Pixel-Powers Graphics System," to appear in *Computer Graphics*, Vol. 20, No. 3, August 1986 (Proc. SIGGRAPH 86).

Kedem, Gershon and John L. Ellis, "Computer Structures for Curve-Solid Classification in Geometric Modeling," Technical Report TR84-37, Microelectronic Center of North Carolina, Research Triangle Park, Sept. 1984.

Poulton, John, Henry Fuchs, John D. Austin, John G. Eyles, Justin Heinecke, Cheng-Hong Hsieh, Jack Goldfeather, Jeff P. Hultquist, and Susan Spach, "PIXEL-PLANES: Building a VLSI-Based Graphic System," *Proceedings of 1985 Chapel Hill Conference on VLSI*, H. Fuchs, ed., Computer Science Press, Rockville, Md.

* This research supported in part by the Defense Advanced Research Projects Agency, contract DAAG29-83-K-0148 (monitored by the US Army Research Office, Research Triangle Park, NC), the National Science Foundation, grant ECS-83-00970, and the Microelectronic Center of North Carolina.

HARDWARE ASSISTANCE FOR Z-BUFFER VISIBLE SURFACE ALGORITHMS

Kellogg S. Booth, David R. Forsey, and Alan W. Paeth

Computer Graphics Laboratory, Department of Computer Science
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1
Tel: (519) 888-4534, E-Mail: KSBooth%watCGL@Waterloo.CSNet

ABSTRACT

The well-known "z-buffer" algorithm for solving the visible surface problem has a number of points in its favor, the main one being that it is amenable to very efficient hardware implementation at little additional cost in many existing frame buffer systems. The traditional software implementation of the algorithm assumes explicit initialization of both the image buffer and the z-buffer before each image is generated. This paper describes a simple technique for synchronizing initialization and image generation so the two can be performed in parallel, allowing complete overlap to be achieved and effectively eliminating the time needed for explicit initialization of the frame buffer. The technique assumes a modest investment in additional hardware within the frame buffer.

RÉSUMÉ

Parmi les algorithmes de surfaces cachées, l'algorithme du "z-buffer" a un certain nombre d'avantages à son actif. Le plus important de ceux-ci étant que cet algorithme est peu coûteux à implémenter au niveau hardware dans les systèmes actuels de "frame-buffer". Les techniques traditionnelles d'implémentation de cet algorithme forcent le logiciel à initialiser explicitement le "z-buffer" et le buffer image avant le transfert de l'image. Cet article décrit une technique simple qui permet de synchroniser l'initialisation et la création de l'image afin qu'elles puissent être réalisées simultanément. Ceci permet d'éliminer le temps perdu lors de l'initialisation du "frame-buffer." Cette technique suppose un faible investissement en matériel additionnel à l'intérieur du "frame-buffer."

Keywords: *double-buffering, frame buffer, real-time, visible surfaces, z-buffer.*

INTRODUCTION

The problem of computing only the *visible surfaces* of a scene has by now been well-studied [12]. One approach that has gained popularity is the *z-buffer* algorithm first described by Catmull [4,5]. With a z-buffer the depth-sort required of a visible surface algorithm is accomplished by maintaining, for each pixel, a record of the *z-depth* of the object whose intensity (color) is stored at that pixel. As subsequent objects are scan converted into the frame buffer, their z-depths are compared and used to decide whether the new object is in front of or behind the object currently displayed at each pixel. In the former case both the intensity and z-depth are changed for the pixel, but in the latter case no action is taken. A complete explanation of the z-buffer algorithm appears in standard text books on computer graphics [7,10].

In the following sections we first define our model of a frame buffer and then look at different ways of adding additional hardware to the frame buffer to speed up the z-buffer algorithm. The first approach almost doubles the amount of memory in the frame buffer and is presented solely to motivate the other two. The second approach adds only a single bit to each pixel but requires a more complicated memory controller that could lead to significant timing problems in the video chain. The third approach adds two more bits to each pixel and hence admits an implementation that requires no additional function in the memory controller beyond what is available in current frame buffers, although a modest change is still required to hardware further down the video chain.

The basic z-buffer algorithm performs well in almost all respects except for considerations of *antialiasing*. This deficiency stems from the fact that the z-buffer maintains depth information on a pixel-by-pixel basis, and thus has no way to discriminate objects at subpixel resolution. This is unfortunate because aliasing artifacts introduced by the scan conversion process can be very objectionable in practice. Some researchers have suggested techniques to incorporate antialiasing strategies into a z-buffer algorithm, but those techniques either require auxiliary storage or

additional processing time beyond the basic scan conversion algorithm, or they fail to achieve the desired level of image quality [3,6,8]. The approaches described in this paper apply to z-buffer algorithms enhanced for antialiasing, although we do not address the issue explicitly. Instead, we adopt the attitude that an important application of the z-buffer technique is to high-performance raster systems for which real-time or quasi-real-time performance is desired and that proper antialiasing is a luxury as yet not consistent with that goal.

We distinguish between the *update rate*, the rate at which new images are computed by the algorithm, from the *refresh rate*, the rate at which the computed images are displayed on the monitor. The update rate is almost always established by limits in processing power and the desire to display complex images and thus tends to remain a bottleneck even as advances are made in hardware and software, whereas the refresh rate is set by the need to overcome flicker effects inherent in the human visual system and can be regarded as a relative constant.

Our definition of *real-time* will be that a complete image is generated within a single refresh cycle (usually 1/30 or 1/60 of a second). The notion of *quasi-real-time* will imply image generation that closely approximates that rate (at worst an update happens every 1/5 of a second). Ron Baecker has already championed the claim that such rates give an acceptable illusion of continuous simulation if the update rate and the refresh rate are suitably synchronized [2]. To be effective, a steady refresh of the current image must be maintained throughout the scan conversion process for the next image. Because of this, we will be interested only in the *double-buffered* version of the algorithm in which a *refresh processor* displays one image while an *update processor* is generating the next image.

With this set of ground rules, we are ready to discuss the performance of the z-buffer algorithm and to examine alternatives to the traditional implementation. The operations performed by the versions of the z-buffer algorithm that we will consider remain largely the same in each of the implementations. The differences lie in the way that the operations are partitioned between the two processors (the update processor and the refresh processor) and in the way that the two processors synchronize their operations.

THE FRAME BUFFER

Our model assumes that a frame buffer contains a large amount of *pixel memory* indexed by two-dimensional (x,y) addresses and that each pixel is divided into fields composed of a number of bits. For our purposes at least three fields are necessary in a pixel. These fields will be designated the I_0 and I_1 fields (two *intensity buffers*, one for the image currently being displayed by the refresh processor and the other

for the image being computed by the update processor) and the Z field (a single *depth buffer*). Frame buffer memory is dual-ported to allow the update processor to read and write pixels (randomly) at the same time that the refresh processor reads pixels at video rates (in scan-line order). The refresh processor passes the pixels down the *video chain* where the color information stored in each pixel is converted to analog signals suitable for display on a monitor, possibly after interpretation by lookup tables or other devices whose function is not important to the present discussion.

Our model for a frame buffer is patterned after the Adage/Ikonas RDS 3000, the hardware on which we have implemented these algorithms. Most of the ideas presented here apply to other frame buffer architectures, although we do assume that the video chain is similar to the control, crossbar, and lookup table modules available in the Ikonas [1,9]. Not all of these features are required for most of the z-buffer techniques, although the final algorithm presented here actually assumes a slightly enhanced crossbar switch over what is supplied by Adage. Our goal is not to restrict attention to a particular frame buffer architecture, but to point toward general hardware features that will substantially enhance the performance of z-buffer algorithms at modest cost.

Double-buffering is accomplished in a frame buffer by the refresh processor reading from I_0 during odd update cycles and from I_1 during even update cycles, thus allowing the update processor to use I_1 and Z for scan conversion during odd update cycles and I_0 and Z during even update cycles. Only one z-depth field is necessary because once the image has been rendered, its z-depths are no longer needed.

The selection of specific fields for reads and writes by both the update processor and the refresh processor may be accomplished by masking and shifting (either in hardware or through a combination of hardware and software) or by using address offset registers when the various fields are stored in different areas of frame buffer memory. These details are not important for the discussion, so we will assume that the frame buffer maintains all of the fields associated with a pixel within a single "word" and that both processors are capable of selecting particular fields with no penalty in time. It is convenient to assume that this is accomplished by *mask registers* and *shift registers*, associated with each processor, that perform selective load and store operations to only those bits indicated by the mask, leaving the other bits of a pixel unchanged.

Any number of bits may be associated with the two intensity buffers. Common configurations use 8 bits (with color lookup tables to achieve a full color space) or 24 bits (8 bits each of red, green and blue). The z-depth must be able to discriminate objects within the scene, so between 8 and 32 bits are commonly assumed. Sutherland and Hodgman discuss a scaling strategy for making the best use of the precision

available [11]. This totals to from 24 to 80 bits per pixel, depending upon the amount of color and z-depth desired. The memory requirement can be reduced somewhat if only the intensity buffer currently being displayed is located within the frame buffer itself (the other fields being stored on the host) but our technique is designed for high-performance systems in which all of the memory resides within the frame buffer to achieve the necessary update and refresh rates. Only the two intensity buffers are accessed by the refresh processor in any of these schemes, so some savings in cost could be achieved by making the z-depth memory single-ported, but this might preclude using the memory for other purposes and is thus a less general architecture.

We will label each update cycle by an integer, but often will only use its even or odd parity (the low-order bit). Thus many places in our algorithms where k is manipulated as an integer modulo some base (usually two) the manipulation can be implemented with simple bit operations such as complementation, rather than with the more expensive increment and modulus operations.

The next three sections present increasingly sophisticated versions of the z-buffer algorithm. The first is the standard implementation, suitable for a very basic frame buffer. The subsequent versions achieve increased performance by partitioning the calculation differently among the update and refresh processor and by using modest hardware assistance to synchronize the calculations.

SOLUTION #1

TRADING OFF MEMORY FOR PROCESSING TIME

The basic appeal of the z-buffer algorithm is that it affords a complete solution to the visible surface algorithm for little more than the time required to perform simple scan conversion without the visible surface calculation. The algorithm is usually implemented entirely in the update processor, with the refresh processing serving only to display the resulting image on the monitor. The two procedures Update#1 and Refresh#1 shown in Figure 1 express the interlocked cycles in a standard implementation of the z-buffer algorithm. Normally the refresh cycle would be implemented entirely in hardware, but we describe it here as if it were implemented in software to provide a uniform presentation of the two processes. The processes are loosely synchronized in this basic version of the z-buffer algorithm. At the start of each update cycle the shared variable k is incremented and this causes the refresh processor to swap image buffers with the update processor. Actual implementations would usually include a provision to further synchronize the double-buffering so that buffers are swapped only at the end of a complete frame or field time.

This is the standard z-buffer algorithm presented in text books and is easily implemented on most frame buffers. The main procedure invokes (once) the setup

```

PROCEDURE Initialize#1;
  k := 0;
  FOR y := maxY DOWNT0 0 DO
    FOR x := 0 TO maxX DO
      I0[x,y] := background;
    OD;
  END Initialize#1;

PROCEDURE Update#1;
  WHILE true DO
    k := k+1;
    FOR y := maxY DOWNT0 0 DO
      FOR x := 0 TO maxX DO
        Ik mod 2[x,y] := background;
        Z[x,y] := ∞;
      OD;
      FOR every object in the scene DO
        FOR every pixel (x,y) in the object DO
          IF (object[x,y].z < Z[x,y]) THEN
            Ik mod 2[x,y] := object[x,y].color;
            Z[x,y] := object[x,y].z;
          FI;
        OD;
      OD;
      { optional wait for next frame }
    OD;
  END Update#1;

PROCEDURE Refresh#1;
  WHILE true DO
    FOR y := maxY DOWNT0 0 DO
      FOR x := 0 TO maxX DO
        display Ik-1 mod 2[x,y];
      OD;
    OD;
  END Refresh#1;

```

Figure 1. The Standard Z-Buffer Algorithm

in Initialize#1 and then invokes (in parallel) the two procedures Update#1 and Refresh#1 which never terminate, but cycle continuously as the double-buffering scheme alternately updates the two image buffers. The refresh processor is merely cycling through memory performing the standard frame buffer readout to the video hardware. If the algorithm is being used to generate a single frame, there is really nothing to discourage its use. But if the algorithm is being used to generate a sequence of frames (as assumed here) the algorithm does not fully utilize the available hardware.

In this situation there is a bottleneck that may potentially degrade performance. The z-buffer algorithm requires that the intensity and z-depth fields be reset to their initial values (background color and

infinity) before each update cycle. The procedure Update#1 performs this initialization explicitly at the beginning of each update cycle. This can be time-consuming for two reasons. The first is that the update processor can only perform initialization when it is not performing scan conversion and thus the full bandwidth of the update processor is not available for scan conversion, an unfortunate consequence because real images frequently require substantially more update time than refresh time.

This is related to the second reason, which is that even update processors with special purpose hardware may not be able to write all of the pixels within the frame buffer in one refresh cycle. The refresh processor reads multiple pixels during a single memory cycle because it looks at pixels in scan-line order and thus can access multiple memory chips in parallel. This allows it to achieve a complete refresh within one frame time. The update processor typically does not do this because it is designed for random access to the frame buffer. The result is that one or more refresh cycles may be "wasted" between successive update cycles while the update processor is busy initializing instead of rendering. This degrades the real-time or quasi-real-time performance of the system by a significant percentage.

Because it accesses multiple pixels during a single memory cycle, the refresh processor is capable of performing the initialization in a single refresh cycle. Some frame buffers support this by allowing the refresh processor to change values in selected fields of pixel memory as each pixel is written back into memory after being read during the refresh cycle [1]. This allows the refresh processor to initialize the second image buffer and the z-depth buffer in a single refresh cycle.

Unfortunately, unless initialization is synchronized with image generation there is little advantage to this approach because the update processor must wait for at least one complete refresh cycle after update cycle k to insure that the refresh processor has completely reset both the I_{k+1} and Z fields before it can begin update cycle $k+1$. This means that the update processor will be idle a significant amount of the time (recall that the slowest update rate for quasi-real-time is 1/5 second so that even with a refresh cycle of 1/60 second the image processor would be idle more than 8% of the time — in the worst case the processor would waste 50% of its bandwidth while maintaining a 1/30 second refresh cycle and a 1/15 second update cycle). The percentage of idle time for the update processor is an important consideration because it determines an upper bound on the complexity of the image that can be rendered. This problem can be overcome by the addition of extra hardware, in this case a substantial increase in frame buffer memory.

SOLUTION #2 TRADING MORE MEMORY FOR PROCESSING TIME

```

PROCEDURE Initialize#2;
  k := 0;
  FOR y := maxY DOWNT0 0 DO
    FOR x := 0 TO maxX DO
      I0[x,y] := background;
      Z0[x,y] := ∞;
    OD;
  END Initialize#2;

PROCEDURE Update#2;
  WHILE true DO
    k := k+1;
    FOR every object in the scene DO
      FOR every pixel (x,y) in the object DO
        IF (object[x,y].x < Zk mod 2[x,y]) THEN
          Ik mod 3[x,y] := object[x,y].color;
          Zk mod 2[x,y] := object[x,y].z;
        FI;
      OD;
    OD;
    { mandatory wait for next frame }
  OD;
END Update#2;

PROCEDURE Refresh#2;
  WHILE true DO
    FOR y := maxY DOWNT0 0 DO
      FOR x := 0 TO maxX DO
        display Ik-1 mod 3[x,y];
        Ik+1 mod 3[x,y] := background;
        Zk+1 mod 2[x,y] := infinity;
      OD;
    OD;
  OD;
END Refresh#2;

```

Figure 2. The Triple-Buffered Z-Buffer Algorithm

To achieve real-time or quasi-realtime performance a third image field I_2 can be added to the frame buffer along with a second z-depth field Z_1 (the original depth field becomes Z_0). In this case the update processor cycles between three image memories, with the refresh processor displaying from I_{k-1} , the update processor writing into I_k , and the refresh processor initializing I_{k+1} (all subscripts for I are now modulo 3 instead of modulo 2). Similarly, the update processor uses Z_k for its visible surface calculation while the refresh processor is initializing Z_{k+1} (these subscripts are modulo 2 since only two depth buffers are required).

As long as each update cycle requires at least one entire refresh cycle (a modest assumption since a faster update rate would imply that the image was being updated faster than it was being viewed on the monitor) the refresh processor will be able to initialize a new

image and depth buffer in time for each update cycle, thus freeing the update processor from any overhead for initialization. Procedures Update#2 and Refresh#2 to accomplish this are straightforward modifications to Update#1 and Refresh#1.

The clear drawback to this scheme is the massive increase in frame buffer memory. The requirement for image memory has increased by 50% (from two buffers to three) and the requirement for depth memory has increased by 100% (from one buffer to two). For the case of a full 24-bit image buffer and a 32-bit depth buffer, this is a total of 136 bits per pixel, an increase of 70%. While this might be appropriate for the increased performance, we are at best getting an improvement that is of the same order of magnitude as the increase in memory cost. We can do much better.

SOLUTION #3 TRADING OFF HARDWARE COMPLEXITY FOR LESS MEMORY

An alternative is to use one additional bit in the frame buffer as a cycle counter (a *dirty bit*) to achieve complete overlap of image generation and initialization while avoiding the necessity of adding an additional set of image and depth buffers. As for Solution #2, this will in fact achieve an update rate that is equal to the refresh rate for simple scenes (something not achievable with the standard software z-buffer algorithm) but at far less hardware cost. We assume that the frame buffer has been extended to include a one-bit field D containing the parity (low-order bit) of k, the update cycle counter.

The frame buffer is initialized once, before the actual z-buffer algorithm begins, so that I_0 is the "background" color and D is 0, the parity of the first image. The setting of I_1 and Z are arbitrary. The update processor begins image generation cycle 1 with the refresh processor initializing both I_1 and Z. When the z-buffer algorithm has generated its first image, the refresh processor begins displaying from I_1 and initializing both I_0 and Z, but it performs the initialization selectively using the cycle number and information kept in the D field of each pixel. The system assumes a steady-state operation in which the two processors synchronize their activity after each update cycle through the shared cycle number and the D field.

This is accomplished in the following way. The update processor proceeds as it normally would, assuming that both I_k and Z have been initialized previously by the refresh processor, even though the refresh processor may not have visited some (or all) of the pixels. The key idea is that each time the update processor performs a depth comparison (testing the z-depth of an object against the value stored in the Z field of a particular pixel) it biases the comparison in favor of the new z-depth (that of the object) if the D field does not match the parity of the update cycle

```

PROCEDURE Initialize#3;
  k := 0;
  FOR y := maxY DOWNT0 0 DO
    FOR x := 0 TO maxX DO
       $I_0[x,y] := \text{background};$ 
      D[x,y] := 0;
    OD;
  END Initialize#3;

PROCEDURE Update#3;
  WHILE true DO
    k := k+1;
    FOR every object in the scene DO
      FOR every pixel (x,y) in the object DO
        IF (D[x,y] = k-1 mod 2)
          OR (object[x,y].x < Z[x,y]) THEN
           $I_{k \bmod 2}[x,y] := \text{object}[x,y].\text{color};$ 
          Z[x,y] := object[x,y].z;
          D[x,y] := k mod 2;
        FI;
      OD;
    OD;
    { mandatory wait for next frame }
  OD;
END Update#3;

PROCEDURE Refresh#3;
  WHILE true DO
    FOR y := maxY DOWNT0 0 DO
      FOR x := 0 TO maxX DO
        display  $I_{k-1 \bmod 2}[x,y];$ 
        IF D[x,y] = k-1 mod 2 THEN
           $I_{k \bmod 2}[x,y] := \text{background};$ 
          Z[x,y] := infinity;
          D[x,y] := k mod 2;
        FI;
      OD;
    OD;
  OD;
END Refresh#3;

```

Figure 3. The Z-Buffer Algorithm Using A Dirty Bit

number. This in effect allows the update processor, by checking the D field, to detect those pixels for which the refresh processor has yet to perform the appropriate initialization and to substitute the value infinity for whatever (incorrect) z-depth appears in the frame buffer. The update processor always re-writes the D field with the parity of the current update cycle number each time it stores into the frame buffer to avoid the problem of the refresh processor mistakenly initializing pixels that have already been used for the current update cycle.

The refresh processor must modify its operation so that it checks the D field before initializing a pixel. Were this not the case, it might overwrite a pixel that the update processor had already computed, since the initialization and update take place simultaneously. The refresh processor only performs an initialization operation if the D field is *not* the same parity as the current update cycle (put another way, it only initializes those pixels whose D fields equal $k-1$). Pixels being initialized have their D fields set to k (not for the benefit of the update processor, but so the refresh processor knows to re-initialize them during the update cycle $k+1$).

For this scheme to work two assumptions must hold. The first is that the refresh processor must be allowed to complete at least one complete cycle between update cycles. As we have already noted, this is a reasonable assumption and is easily guaranteed by a simple test performed at the start of each frame. The second assumption is that the refresh processor makes its access to memory in a single atomic operation ("read-modify-write"). If this were not the case, the refresh processor might overwrite a pixel whose D field changed between the time that it was read and the time that it was written back to memory. The implication of this assumption is that the refresh processor must have a reasonably sophisticated interface to frame buffer memory — it is fetching multiple pixels in parallel, all of which must have their D fields checked and their I and Z fields modified in a single memory cycle.

The procedures `Update#3` and `Refresh#3` indicate the z-buffer algorithm using the D field to overlap initialization by the refresh processor with image generation by the update processor. The algorithm as stated assumes that the update processor only begins a new cycle during the start of a new frame. This can be weakened significantly to the requirement that the update processor not begin a new cycle until the refresh processor has performed at least one complete refresh cycle since the last update cycle began (our standard assumption). It may also be desirable to insist that the refresh processor not change its value of k except at the beginning of a frame (or at least a field) because of disturbing video effects.

Before continuing, the reader may want to verify that the algorithm works as stated, with no race conditions existing that depend on the order in which objects are scan converted by the update processor or the order in which pixels are initialized by the refresh processor. In doing so, special note should be made of the assumption that the refresh processor's memory accesses are atomic.

The cost in additional memory for this scheme is minimal. Only one extra bit is needed at each pixel. The increased sophistication in the refresh processor, however, is more substantial and may push up the cost of the video hardware significantly. Instead of performing a simple read-modify-write cycle (as it would for Solution #2) in which the new values written

```

PROCEDURE Initialize#4;
  k := 0;
  FOR y := maxY DOWNT0 0 DO
    FOR x := 0 TO maxX DO
      D0 := false;
      D1 := false;
    OD;
  END Initialize#4;

PROCEDURE Update#4;
  WHILE true DO
    k := k+1;
    FOR every object in the scene DO
      FOR every pixel (x,y) in the object DO
        IF (NOT Dk mod 3[x,y])
          OR (object[x,y].x < Z[x,y]) THEN
          Ik mod 2[x,y] := object[x,y].color;
          Z[x,y] := object[x,y].z;
          Dk mod 3[x,y] := true;
        FI;
      OD;
    OD;
    { mandatory wait for next frame }
  OD;
END Update#4;

PROCEDURE Refresh#4;
  WHILE true DO
    FOR y := maxY DOWNT0 0 DO
      FOR x := 0 TO maxX DO
        IF Dk-1 mod 3[x,y] THEN
          display Ik-1 mod 2[x,y]
        ELSE
          display background;
          Dk+1 mod 3[x,y] := false;
        FI;
      OD;
    OD;
  OD;
END Refresh#4;

```

Figure 4. The Z-Buffer Algorithm Using 3 Dirty Bits

back to memory are independent of those read from memory (at least for the fields that change) the refresh processor must now check the status of the D field (a single bit) to determine whether the original contents are to be left in the $I_{k \bmod 2}$ and Z fields or if they are to receive initialization values. All of this must be performed in parallel for anywhere from 16 to 64 pixels, depending upon the design of the frame buffer memory interface. We can avoid this necessity, while still retaining the performance, by adding a few more bits to each pixel.

SOLUTION #4 TRADING BACK SOME OF THE MEMORY FOR HARDWARE SIMPLICITY

The reason the refresh processor's memory controller must be so complicated is that it must decide (very rapidly) which pixels must have their I and Z values modified. This dependence of certain bits within the pixel on other bits within the pixel may be difficult to determine, especially if the large pixel size requires some of the bits to reside on different boards. The solution proposed is to eliminate the necessity of checking the current pixel contents before deciding what to write back into memory during the refresh cycle. What we want is an *oblivious* memory controller, one which always writes the same pattern (or at least one which always changes the same bits to the same values) independent of the current pixel contents.

The trick is to use *three* dirty bits, one for each of the cycles $k-1$, k , and $k+1$. These can then be administered independently by the refresh controller. If we interpret D_k as a Boolean value (true or false) that tells whether the current pixel contents have been set during the corresponding update cycle, the job of the refresh processor becomes much simpler. During steady state, the refresh processor will be fetching pixels from I_{k-1} and initializing D_{k+1} while the update processor is modifying D_k and Z.

The only catch to this scheme is that pixels that are never rendered by the update processor (because they correspond to background) will remain uninitialized in their I and Z fields. This is not a problem if the refresh processor interprets the D field before passing pixel values on to the rest of the video chain. It must simply check D_{k-1} and if it is false (meaning that this pixel was never set during the previous update cycle) then it should pass on background color rather than what is stored in I_{k-1} . Hardware to perform this task is much simpler than the massive parallel checking required for Solution #2 because it can be performed on a pixel-by-pixel basis using techniques similar to lookup tables.

It is interesting to note that three dirty bits are necessary to implement this scheme. Two are not enough. D_{k-1} must remain untouched during update cycle k or else the refresh processor will become confused as to what is or is not background. D_k obviously must be initialized before update cycle k begins and cannot be changed except by the update processor. Neither field is free for initialization during update cycle k . Thus a third dirty bit D_{k+1} is required.

CURRENT IMPLEMENTATIONS

These algorithms are implemented on the Adage/Ikonas RDS 3000. Solution #1 is the standard z-buffer algorithm. The only change in the implementation from what has been presented is that

the Z field is typically stored in off-screen pixel memory because the frame buffer has only 32 bits per pixel and 24 are used for intensity. On frame buffers with no off-screen memory the entire algorithm can be accommodated on-screen by decreasing the number of bits allocated to intensity and setting the lookup tables appropriately to ignore bits in the Z field as they are read out during display.

Solution #2 (the triple-buffered version) has also been implemented on the Ikonas, but with only limited intensity and z-depth information due to the requirement that all of the fields reside in on-screen memory. This stems from the fact that the *auto-clear* feature of the Ikonas (which writes zeros into memory during the refresh cycle, subject to a write mask that determines the bits in a pixel to be cleared) only processes visible pixels. Adopting conventions for intensity and z-depth that encode the background color and the maximum z-depth as zero allows existing hardware to handle the initialization in the refresh processor. A more natural encoding is possible if the auto-clear feature uses the shading registers (available on with the Ikonas GM memory boards) to set the values to be written into memory, rather than always writing zeros into the fields specified by the write mask registers. Buffer swapping is accomplished by manipulating the crossbar switch and the lookup tables.

Solution #3 (the dirty bit) is not directly implementable on the Ikonas because the auto-clear feature has no way of conditionally modifying bit fields. This is symptomatic of the objection raised earlier that this approach assumes more intelligence in the refresh processor's memory controller than is likely to exist in a frame buffer.

Solution #4 (three dirty bits) is easily implemented on the Ikonas using the convention that true is a 0 bit and false is a 1 bit. The auto-clear feature is used to initialize the dirty bits during refresh and a combination of the crossbar switch, the lookup tables, and the overlay option is used to modify the pixel readout to background color for pixels whose values have not been set by the update cycle. (The overlay option on the Ikonas allows certain bits — the appropriate dirty bit in our case — to select an alternate lookup table if the bits are non-zero. By setting all of the entries in the alternate lookup table to be the background color the correct modification is performed as pixels are read from memory during refresh.)

FURTHER CONSIDERATIONS

There is a question as to how dynamic the allocation of the various fields should be within a pixel. On the update processor all of the field selection can be fairly easily accomplished using masking and shifting. If the intensity and z-depth fields are multiples of eight bits, simple byte swapping logic can be used to present

an interface to the processor that is independent of the exact field being selected (i.e., the update processor will always present intensity of z-depth values as right-adjusted 32-bit values that will be automatically stored in the correct fields of a pixel). The dirty bits may have to be handled as special cases, although choosing appropriate conventions such as forcing all z-depths to be positive (and thus saving the sign bit for the dirty bit) could be employed. Generalized crossbar switches, similar to that on the Adage/Ikonas RDS 3000, used by both the update and refresh processors would clearly solve any efficiency issues related to problem.

In implementing such hardware, some *caveats* are in order. Any registers that allow values or fields to be selected should be both writable and readable. If necessary, there should be shadow registers so the values can be read back, although it is preferable that the registers themselves be readable directly.

It is worth noting that only the refresh circuitry needs a read-modify-write cycle that is atomic for the schemes to work. This is an important consideration, especially if the update processor is composed of distributed or pipelined tilers that separate their read accesses from their write accesses by multiple cycles. Such architectures are particularly useful for z-buffer algorithms because they increase parallelism and thus the update rate and yet require little or no synchronization among the multiple update processors because the z-buffer algorithm is itself inherently distributed. Our algorithms will not suffer in this case.

ACKNOWLEDGEMENTS

Schemes similar to those presented here have been implemented in custom hardware by Trillium Corporation in their flight simulator systems [13].

A lucid discussion of the issues involved in building the memory controller for the refresh processor is given in Whitton's excellent article on frame buffer design [14]. The crossbar switch for the Adage/Ikonas RDS 3000 was originally suggested by Henry Fuchs.

This article would not have been written except for the encouragement of Marcell Wein. Numerous discussions with Nick England and Mary Whitton have contributed to our appreciation of the Ikonas architecture and its versatility. Alain Brossard kindly provided the translation for the résumé. The research reported here was supported by the Natural Sciences and Engineering Research Council of Canada under a variety of grants.

REFERENCES

[1] Adage, "RDS 3000 Manual."
[2] R. M. Baecker, "Digital Video Display Systems and Dynamic Graphics," *Computer Graphics* 17:3, (August, 1979) pp. 48-56.

[3] L. Carpenter "The A-Buffer, An Antialiased Hidden Surface Method," *Computer Graphics* 18:3, (July, 1984) pp. 103-108.
[4] E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, (doctoral dissertation) Technical Report UTEC-CSc-74-133, Department of Computer Science, University of Utah (December, 1974).
[5] E. Catmull, "Computer Display of Curved Surfaces," *Proceedings IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures*, (May, 1975), reprinted in *Tutorial and Selected Readings in Interactive Computer Graphics*, H. Freeman (ed.), IEEE Computer Society (1980) pp. 309-315.
[6] K. D. Evans "An Approximate Method for Anti-Aliasing, Using a Random Access A-Buffer," *Proceedings Graphics Interface '84*, (May, 1984) p. 109.
[7] J. D. Foley and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley (1982).
[8] E. Fiume, A. Fournier, and L. Rudolph, "A Parallel Scan Conversion Algorithm with Anti-Aliasing for a General-Purpose Ultracomputer," *Computer Graphics* 17:3, (July, 1983) pp. 141-150.
[9] S. A. Mackay and K. S. Booth, "Techniques for Frame Buffer Animation," *Proceedings Graphics Interface '82* (May, 1982) pp. 213-220.
[10] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, second edition, McGraw-Hill (1979).
[11] I. E. Sutherland and G. W. Hodgman, "Reentrant Polygon Clipping," *Communications of the ACM*, 17:1, (January, 1974) pp. 32-42.
[12] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *Computer Surveys*, 6:1 (March, 1974) pp.
[13] R. Swallow, personal communication.
[14] M. Whitton, "Memory Design for Raster Graphics Displays," *IEEE Computer Graphics and Applications*, 4:3, (March 1984) pp. 48-64.

Eliminating the Dichotomy Between Scripting and Interaction

John F. Schlag

Three Dimensional Animation Systems Group
Computer Graphics Laboratory
New York Institute of Technology

Abstract

Throughout the computer graphics literature, but in modeling and animation lore in particular, the prevalent attitude seems to be that scripting and interaction are irreconcilably different types of user interface. This paper propounds the belief that this dichotomy is a myth and gives examples of systems which encourage this belief. Two of the systems described are a keyframe animation program and a geometric modeling program developed at the NYIT Computer Graphics Laboratory. Both of these systems are used on a daily basis for production, development and research.

1. Introduction

Scripting and interaction are widely used interface styles for modeling and animation programs. The advantages and disadvantages of each are well known. Scripting systems allow command sequences to be incrementally modified and reexecuted, but allow no interactive input of data and operations. Interactive systems allow this input, but lose all record of commands previously executed, leaving the user to start over if some intermediate parameter affecting the final result needs to be changed. In almost every publication describing a new modeling or animation system, there is a paragraph or two which notes these properties and proceeds to ballyhoo the advantages of the method chosen. As with many other such apparent dichotomies, most designers accept without question the division, pick one technique, implement it and resign themselves to writing the obligatory text. From this lamentable situation we may conclude that the production of a user interface which integrates scripting and interaction is *difficult*, but certainly not that it is impossible.

The integration of scripting and interaction is worthwhile for several reasons. The first is that a more general conceptualization of user interfaces results. This has the potential for expanding the set of problems that can be solved with a computer. On a more practical note, interactive graphics workstations are often expensive and therefore scarce. In many graphics houses, the time available on such resources is a limiting factor in the amount and/or complexity of animation producible. It helps

greatly if users can work, albeit more slowly, at ordinary terminals, using scripting instead of interaction.

Resource limits are not the only reason to use scripting at one workstation and interaction at another. The production of computer animation requires many diverse activities, most of which are accomplished more efficiently with one technique than the other. Some establishments may boast interactive *and* scripting tools for particular activities, but rarely do these tools interface to a common database format. More often, the systems are incompatible "competitors", perhaps having been written by different individuals, and any attempt to use both to solve a single problem is frustrated by the need to reorganize data, shuffle files about and translate between formats. A system which integrates scripting and interaction can be used for conceptually diverse activities in whatever mode is appropriate. A banner example of the type of production which needs this is the now famous robot ant animation by Lundin [15], in which the basic 3d path of the model is supplied interactively and the dynamics are supplied by an explicit computational model.

For perspective, the more general problem here is the development of *editing theory*. A general theory of editing should be independent of the data to be edited, so it should be applicable not only to ordinary text editing, but to other activities such as geometric modeling (shape editing), animation and robot programming (motion editing), music production (sound editing), painting (image editing) and computer programming (algorithm editing). What we would like for any given data format is a logical and complete set of functions for manipulating that format, and both interactive and scripting interfaces to these functions.

When differentiating between interaction and scripting, it is natural at some point to quibble about exactly what *interactive* means. Some would argue that if the time around the typical edit-process-view loop of a scripting system is short enough, the system is interactive. Is there some threshold on this loop time which must be met? "Real" time? Historically, the term has been used to mark a contrast with *batch* systems, where the sizes of input and output data sets are usually large. Systems that produce visible results after small amounts of input

(after every keystroke, say, in a screen editor) we classify as being definitely interactive. An often cited parameter for language-based systems is whether the system is interpretive or compiling. Interpretive systems, however, are rarely used without large application dependent scripts, while compilation systems can call themselves "interactive" simply by connecting their inputs to the keyboard. Some systems [13, 16] blur the distinction even further by loading compiled code into interpretive front ends at run time. The answer to all this, of course, is that the set of interactive systems is a fuzzy one, with no clear dividing lines along any of these axes. Hence, we discontinue our quibbling here.

2. Combining Scripting with Interaction

This section describes the requirements for an integrated scripting/interactive user interface. Several programs/systems which implement these requirements in varying degrees of completeness are drawn upon for examples. *Emacs* is an "extensible, customizable, self-documenting" screen editor developed at MIT [1]. The Unix¹ implementation with which the author is familiar [2] supports extension through an interpreter for a lisp-like language. *Troff* [5] (or *Scribe* or *TeX*, if you prefer) is a typesetting program which supports macros. The *Macintosh* personal computer [17], with its attendant firmware, is the latest paragon of user interface style. *Em* is an animation program written at NYIT for interactive motion specification (animation) of parameterized models [8]. It is unique among these example systems in its use of a large (rather much larger than that of C [4]) formal grammar [7] for defining its basic input language. *Gem* [16] is an interactive geometric modeling program, also written at NYIT, and is the current object of this research.

To develop a model of a user interface which integrates scripting and interaction, one can think either about adding an interactive front end to a scripting system, or about adding scripting features to an interactive program kernel. Both these approaches have merit. The former is more likely to result in a consistent, well-designed system, since the basis for the system is presumably a well-designed language. The latter is more likely to result in a truly flexible system, since the user interface design would proceed unfettered by syntax. *Gem* was developed in the latter mode, and this paper attempts to relate the experience gained from that activity. What, then, is required of an interactive system which also supports scripting?

(1) The first and most obvious requirement is that there must be some appropriate script representation. Text is an obvious choice, but the problem with text is that it forces linear formatting. This may be acceptable for document and music production, but for modeling and animation, where instancing is a way of life, a graphic dataflow format may be more appropriate. (In fact, the frustration of ordinary music notation is due in part to its limited support for instancing.) Even if a linear format is

tolerable, a more cogent objection to text is that it represents the script at the wrong level. Dealing with the script directly as text ignores the higher-level command structure.

There must be a script representation of every interactive editing command, and there must be interactive access to every script command. This brings up the matter of what is and what isn't script. It is important to differentiate commands to an interactive front end from commands that manipulate the data to be edited. The interactive interface provides an *additional*, rather than an *equivalent* means of access to editing commands. This seems a sensible separation, as no one wants to type in reams of text for low level events like tablet movements anyway. In fact, it is useful to be able to siphon off the raw event stream into a file for testing and demonstration purposes (as the Macintosh can do), but there is no need for that file format to be integrated with the script representation.

(2) The system must maintain some internal representation of the script. Unix Emacs uses three representations for script material, one the lisp-like syntax mentioned earlier for defining functions, the second a buffer of raw keystrokes for defining "macros" and the third a private internal format for supporting 'undo'. Em begins by parsing a script with a yacc-generated parser [6], generating a dependency tree and display segments as semantic results. Instead of converting the script to a syntax tree representation, however (as a compiler would for code generation), the original text is stored in memory for reparsing during the interactive updating of parameter values.

(3) The system must provide a way to execute a script. This is the easy part, as pushing the current input source on a stack and reading a script from a file or memory is simple on reasonable operating systems. The only added wrinkle is that scripts should be able to turn to an absolute source (typically the user) for input. In modeling, for instance, a user may want to create a script segment that creates an instance of a highly parameterized primitive by filling in some parameters on its own and prompting for the rest. Emacs provides an assortment of functions for getting values of various types from the keyboard.

(4) The system must be able to create a script from interactive input. Emacs saves raw keystrokes for macro definition; function definition from interactive input is achieved by mapping keystrokes through a *keymap* which associates functions with keys, and then appending the names of the functions to the script. The Synclavier [21] provides a facility for "reverse-compiling" a real-time keyboard performance into a score.

Creating scripts from interactive input is where the real subtleties begin to arise. For example, most scripted operations will be used as subroutines would be in an ordinary programming language. (This restriction is reasonable, since in many languages everything is part of some

subroutine.) The context in which this "subroutine" is to be executed will dictate end conditions for the recording of the script. One solution, used by Emacs for both function and macro definition, is to provide two special interactive commands to start and stop the translation of interactive input into script. These commands must be treated specially in all contexts, as their meaning in a script is unclear at best, and they should not be translated into script when given interactively.

Some decision must be made as to whether the input being translated is to be executed at the time of recording. A good solution to this is to execute the input when it's coming in interactively, and to simply store it away when it's not. Emacs, being "interactive", executes its input while translating it, which makes clear the effects the script will have when executed later. At the other end of the spectrum, Troff defines very clearly a *copy mode* through which input destined for macros is read.

Above, it was maintained that a direct translation of the low level event stream is not an appropriate script representation. The open question is this: if raw events aren't going to be represented directly in the script, how are they going to be represented? High bandwidth events such as tablet and dial movements present a formidable data pollution problem, so some means of compressing these events into a compact interpretation needs to be found.

(5) The system must provide a way to edit a script. This is important, since very little time is spent creating scripts as opposed to editing them (witness software production). For this function, Emacs "cheats" and uses itself to do the editing. It is, after all, a text editor, and its only events are characters from the keyboard. Interactive modeling and animation programs which employ an arsenal of interactive devices are not so fortunate. More special commands are needed here to position the "cursor" where translated script will be inserted, and to delete script elements. Em uses a compromise solution to allow the user to edit the set of current *input modes* (relationships between logical device values and parameter values), which is to edit a text representation of the input modes with a text editor and then read them back in. The text representation is actually a subset of the command language, and the same parser is used for it that is used for the input script.

The support of editing is the most difficult part of integrating scripting and interaction. As an example, consider the support necessary for the special case of 'undo'. At any point in the command sequence, the user can issue an undo command which undoes the previous editing command, and sequences of undo commands have cumulative effect. (An interesting side issue is whether the undo commands should be part of the script, that is, whether you should be able to undo the undoing.) There seem to be only two alternatives for supporting this functionality: either all the operators must be invertible, or the entire state of the system must be snapshotted after every command, both fairly daunting propositions.

In the more general case, commands at any point in the script may be changed. Assuming that an interactive system wants to keep up to date versions of the final results on display, this means that to minimize script reexecution time, every intermediate result must be saved. This generates an obvious conflict with storage requirements. Faced with this, one gives up and accepts the difference between compiled and uncompiled forms of the data, as well as script reevaluation time.

3. An Implementation

This methodology is being explicitly applied in the design of the user interface to Gem, to our geometric modeling program at NYIT. This program sports the usual socially acceptable features such as on-screen menuing, overlapping windows (that support vector data), icons and on-line help, as well as some more unique features such as run-time loading of compiled object code and a rich symbol table structure. Modeling operations include polygon digitizing, translational and rotational sweeping, mirroring, stellation, truncation, boolean set operations, offsetting and quadratic, bicubic and geodesic subdivision [9, 10]. Output databases are produced for our animation and rendering software.

The modeler is extensible at two different levels. The scripting facility described below provides a means for ordinary users to write new functions in the command language of the modeler. The menu structure is dynamically modifiable, so these functions can be inserted at appropriate places or just executed from the keyboard. The run-time loader and symbol table structure are used by developers and application programmers to extend the program in the base programming language. After a run-time package has been loaded in, the program appears for all intents and purposes as it would have if the package had been linked in by the system maintainers before releasing it.

Scripts are represented both externally and internally as text. Execution is simple, given the i/o redirection facilities of Unix [3]. The problem of turning to the user for direct input has been solved by tagging the data rather than calling a special routine. (In fact there is no explicit syntax in the input language for procedure calls.) When interactive input is desired in a script, a special token is inserted which is understood by all the data collecting routines to mean: "ignore this token, push the current input channel on the stack and use the interactive input channel." (This is similar to the method used by the Unix command interpreter [11], which, alas, also does not support subroutines.)

Script generation is accomplished by sending strings to a script from a small number of places in the program. The names of functions are emitted just prior to invocation. When functions need arguments (integers, strings, parts, etc.) they call 'get' functions which prompt for and return a variable of the appropriate type. These functions send the argument values obtained to the script just before returning. This scheme limits the number of places

in the program that have to know about script generation to one per data type.

Because of Gem's lack of a formal grammar to define the input language, a practical problem in the creation of scripts is the difficulty of determining where a macro ends. When the *define-macro* command is given, the translation of events into script is enabled. When this command occurs in a script, commands are not executed during recording, but consumed by the *define-macro* command until it sees the *end-macro* command. In this mode, Gem can be confused if the *end-macro* command appears inside the macro. When the input is interactive, commands are executed during recording. This method does not become confused since the commands consume their arguments.

One of Gem's user interface features which complicates script generation is its use of cancelable commands. Each of the get functions provides a 'cancel' option which causes the calling function to abort. To handle this correctly, the generation of script must be deferred until commands execute to completion. Presently, Gem makes no attempt to handle this problem.

Another interesting issue in script generation is how to handle modelessness. Gem makes an attempt to be as modeless as possible. In the get functions, any input which isn't of the type needed is passed through to a recursive invocation of the command interpreter. For example, while selecting a part, the user can move the camera, make new parts, change the windows around, access the on-line help database and so on. This is one case in which the lack of a formal grammar actually makes the script generation easier: Gem simply prints script commands as they are evaluated. For example:

```
intersect-parts part1 move-camera part2
```

In contrast, if the script format were Lisp, for example, some means would have to be found of splicing in the extra commands:

```
(intersect-parts part1 (progn (move-camera) part2))
```

The conversion of tablet and dial events to script is dependent on context. For example, in several contexts tablet picks are used to select a current element (solid, surface, polygon, edge, vertex, etc.). In these contexts, the tablet pick is considered to be an alias for the appropriate 'select' command, so the script result is the name of the command plus the name of the object selected. Tablet and dial movements, the really high bandwidth events, are dealt with by considering them to be inputs to fancy 'get' functions. A common use for these events, for example, is in the construction of positioning matrices. When the function which collects these events into a matrix returns, it emits an appropriate script representation just like the other 'get' functions.

Two approaches to script editing have been developed.

The first is the compromise adopted by Emacs and Em: dump the script into a file, invoke a text editor on the file and read the file back in. This was implemented as a first cut because it was easy. It also has the advantage of being editor-independent.

The more ambitious script editing scheme, still under development, is to open a text editor as a concurrent process. The editor provides two windows, one an interactive channel to the normal terminal i/o of the modeler process, and the other a buffer for the script to be edited. Script emitted from the modeler is sent to the editor's script buffer and inserted at the current cursor location of that buffer. Hence, the commands of the text editor are available in the script window and the commands of the modeler are available in the other. The drawbacks of this approach are that all the terminal data for the modeler must go through the editor (a performance issue), and, more importantly, only one editor (namely Emacs) can support this mode of operation. If the window system were below the process level, things would be much easier, but unfortunately, our window system is part of the modeler process.

4. Discussion and Conclusions

Two major improvements are planned for the Gem user interface. The first is to reimplement the upper level of the interface in Lisp [12]. This would eliminate many of the problems that arise because of the lack of a formal grammar. The interactive command interface would stay the same, but scripting would be done in Lisp. This has only recently become practical due to the development of a programming environment [14] which supports both C and Lisp, and guarantees that the conversion can be done incrementally, without discarding the several tens of thousands of lines of existing modeling code.

The second improvement being considered is a provision for user-configurable device interaction via function networks [24]. A function network package has been implemented and tested as a run-time package but has not yet been bound into the program. The use of function networks at the modeling level as well would provide a means of representing the dataflow aspects of a model explicitly. This would also make the storage of intermediate results possible, which, as mentioned earlier, opens the door to the optimization of script execution time, providing one is prepared to pay the price in storage requirements.

The objections raised earlier to text as a script representation were based in part on the limited intelligence of current text editors. This objection could possibly be eliminated through the use of syntax-directed editing, a technique which has been under development by the programming language community for some time. The script form would be text, but the text editor would know the syntax of the language so its elements could be handled at a reasonable level.

As conjectured in the introduction, the implemen-

tation of a user interface kernel which provides integrated support for scripting and interaction has proven difficult, but not impossible. The work done so far has begun to provide the insights necessary for the implementation of a truly useful tool for animation production, and the methodology developed seems applicable to a wide range of data editing problems.

5. Addendum: Other Potential Applications

In addition to modeling and animation for computer graphics, another application area that stands to benefit greatly from the integration of scripting and interaction is music production. Composers should be able to create a score in some reasonable frequency content versus time notation (using either traditional staves or some alternative notation [18]), play the score through some instrument, play the instrument to generate a score, and finally edit the score using either instrumental or scripted input. In fact, a first cut at a system with these capabilities can be assembled with current technology [19, 20, 21, 22].

Another area that uses hybrid scripting/interaction techniques already is robot programming. Unimation Puma series robots [23], for example, come with a programming language for scripting and a manual control box for interactively positioning the robot in key configurations. Once a configuration is set, a line of text can be inserted in the program buffer with a press of a button on the manual control. Such a programming environment could be much more generally useful if fleshed out with the remainder of the capabilities described in this paper.

Acknowledgements

Pat Hanrahan started the Gem project, is directly responsible for most of the geometric functionality, and directly or indirectly responsible for large parts of the user interface. Many others have contributed ideas and/or code to the Gem user interface, among them Dan Hopen, Robert McDermott, Peter Oppenheimer, Michael O'Rourke, Jean-Louis Schulmann, Jacques Stroweis and David Sturman. Em was written by Pat Hanrahan and David Sturman. I thank John Lewis for much discussion, goading, critical reading and the C/Lisp environment, and Kurt Fleischer and Steve Rubin of Schlumberger Palo Alto Research for general discussions on scripting vs. interaction and dataflow techniques in modeling.

References

1. *Emacs: The Extensible, Customizable, Self-Documenting Display Editor*, Richard M. Stallman, MIT AIM-519A, March 1981.
2. *UniPress Emacs Screen Editor Manual*, UniPress Software, Edison, NJ.
3. *The UNIX Time Sharing System*, Dennis M. Ritchie and Ken Thompson, Comm. ACM, July, 1974 (republished Jan 1983).
4. *The C Programming Language*, Kernighan and Ritchie, Prentice Hall, 1978.

5. *Nroff/Troff User's Manual*, J. F. Ossana, Unix Programmer's Manual, Volume II.
6. *Yacc: Yet Another Compiler Compiler*, Steven C. Johnson, Unix Programmer's Manual, Volume II.
7. *Mint User's Manual*, David J. Sturman, NYIT internal documentation, September 1985.
8. *Interactive Animation of Parametric Models*, Pat Hanrahan and David Sturman, Siggraph Tutorial #9: In-
9. *A Subdivision Algorithm for Smoothing Down Irregular Shaped Polyhedrons*, D. W. H. Doo, Proc. Interactive Techniques in Computer Aided Design, 1978.
10. *Recursively Generated B-spline Surfaces on Arbitrary Topological Meshes*, Catmull and Clark, Computer Aided Design, Nov. 1978.
11. *An Introduction to the C Shell*, William Joy, 4.2BSD Unix Programmer's Manual, Volume IIc.
12. *Lisp*, Winston and Horn, Addison Wesley, 1981.
13. *The Franz Lisp Manual*, John K. Foderaro, Keith L. Sklower and Kevin Layer, 4.2BSD Unix Programmer's Manual, Volume IIc.
14. *Zlisp*, John Lewis, NYIT internal documentation, March 1985.
15. *Motion Simulation*, Richard V. Lundin, Proc. Nicograph-84.
16. *Gem User's Manual*, John F. Schlag, NYIT internal documentation (in preparation).
17. *Inside Macintosh*, Apple Computer Corp.
18. *MusicWorks*, Hayden Software, Lowell, MA.
19. *Yamaha Digital Programmable Algorithm Synthesizer Operation Manual*, Buena Park, Yamaha International Corp., 1984.
20. *DX7*, Yasuhiko Fukuda, Amsco Publications, London, England, 1985.
21. *Synclavier II Instruction Manual*, New England Digital Corp.
22. *MIDI Specification 1.0*, International MIDI Assoc., N. Hollywood, CA, 1983.
23. *Unimate Puma Mark-II Robot 500 Series Equipment and Programming Manual*, Unimation, Inc., Danbury CT, April 1984.
24. *PS300 Programmer's Reference*, Evans and Sutherland Computer Corp., Salt Lake City, Utah.

¹Unix is a trademark of AT&T Bell Laboratories. Scribe is a trademark of Unilogic, Inc. Macintosh is not a trademark of Apple Computer, Inc.

SURVEY OF TEXTURE MAPPING

Paul S. Heckbert

Computer Graphics Lab
New York Institute of Technology †

ABSTRACT

Texture mapping is one of the most successful new techniques in high quality image synthesis. Its use can enhance the visual richness of raster scan images immensely while entailing only a relatively small increase in computation. The technique has been applied to a number of surface attributes: surface color, surface normal, specularity, transparency, illumination, and surface displacement, to name a few. Although the list is potentially endless, the techniques of texture mapping are essentially the same in all cases. We will survey the fundamentals of texture mapping, which can be split into two topics: the geometric mapping which warps a texture onto a surface, and the filtering which is necessary in order to avoid aliasing. An extensive bibliography is included.

KEYWORDS: texture mapping, texture filter, space variant filter, antialiasing.

INTRODUCTION

Why Map Texture?

In the quest for more realistic imagery, one of the most frequent criticisms of early synthesized raster images was the extreme smoothness of surfaces - they showed no texture, bumps, scratches, dirt, or fingerprints. Realism demands complexity, or at least the appearance of complexity. Texture mapping is a relatively efficient means to create the appearance of complexity without the tedium of modeling and rendering every 3-D detail of a surface.

The study of texture mapping is valuable because its methods are applicable throughout computer graphics and image processing. Geometric mappings are relevant to the modeling of parametric surfaces in CAD and to general 2-D image distortions for image restoration and artistic uses. The study of texture filters leads into the development of space variant filters, which are useful for image processing, artistic effects, depth-of-field simulation, and motion blur.

Definitions

We define a *texture* rather loosely: it can be either a texture in the usual sense (e.g. cloth, wood, gravel) - a detailed pattern which is repeated many times to tile the plane, or more generally, a multidimensional image which is mapped to a multidimensional space. The latter definition encompasses non-tiling images such as billboards and paintings.

Texture mapping means the mapping of a function onto a surface in 3-D. The domain of the function can be one, two, or three-dimensional, and it can be represented either by an array or by a mathematical function. For example, a 1-D texture can simulate rock strata; a 2-D texture can represent waves, vegetation [Nor82], or surface bumps [Per84]; a 3-D texture can represent clouds [Gar85], wood [Pea85], or marble [Per85a]. For our purposes textures will usually be 2-D arrays.

The source image (*texture*) is mapped onto a surface in 3-D *object space* which is then mapped to the destination image (*screen*) by the viewing projection. Texture space is labeled (u, v) , object space is (x_o, y_o, z_o) , and screen space is (x, y) .

We assume the reader is familiar with the terminology of 3-D raster graphics and the issues of antialiasing [Rog85], [Fol82].

Uses for Texture Mapping

The possible uses for mapped texture are myriad. Some of the parameters which have been texture mapped to date are: surface color (the most common use) [Cat74], specular reflection [Bli76], normal vector perturbation ("bump mapping") [Bli78a], specularity (the glossiness coefficient) [Bli78b], transparency [Smi79], diffuse reflection [Mil84], surface displacement and mixing coefficients [Coo84b].

Illumination Mapping

Mapping specular and diffuse reflection is rather different from mapping other parameters, since these maps are not associated with a particular object in the scene, but to an imaginary infinite radius sphere, cylinder, or cube surrounding the scene [Gre86a]. Whereas standard texture maps are indexed by the surface parameters u and v , a specular reflection map is indexed by the reflected ray direction [Bli76] and the diffuse reflection map is indexed by the surface normal direction [Mil84]. The technique can be generalized for transparency as well, indexing by the refracted ray direction [Kay79]. In the special case that all surfaces have the same reflectance and they are viewed orthographically the total reflected intensity is a function of surface orientation only, so the diffuse and specular maps can be merged into one [Hor81].

Illumination mapping, as these techniques are called, facilitates the simulation of complex lighting environments, since the time required to shade a point is independent of the number of light sources. Other reasons for its recent popularity are: it is one of the few demonstrated techniques for antialiasing highlights [Wil83], it is an inexpensive approximation to ray tracing for mirror reflection, and to radiosity methods [Gor84] for diffuse reflection of objects in the environment. Efficient filtering is especially important for illumination mapping, where variations in surface curvature often necessitate broad areas of the sky to be averaged.

Since specular reflection varies as a function of the viewing direction, it is most conveniently computed on the fly, as in ray tracing. Diffuse reflection of the environment, however, has not yielded to ray tracing even when stochastic methods [Coo84a] are used. The problem is that diffuse reflection scatters light over an entire hemisphere, not a narrow cone, as does specular reflection. Fortunately diffuse reflection is independent of viewing direction, so the incident illumination at each surface point can be precomputed and treated as a texture [Coo84b]. Previous methods have approximated this using polygon subdivision to model hard shadows [Ath78], soft shadows [Nis83], beams of light [Hec84], or indirect illumination [Gor84]. With the development of more efficient algorithms for its computation, incident illumination promises to be a common use for textures in the future.

Even when direct support for illumination mapping is unavailable, tricks can be employed which give a visually acceptable approximation. Rather than calculate the exact ray direction at each pixel, one can compute the reflected or refracted ray direction only at polygon vertices and interpolate it, in the form of u and v texture indices, across the polygon using standard methods. This approximation is similar to that made by beam tracing [Hec84].

† Current address: Pacific Data Images, 1111 Karlstad Dr., Sunnyvale, CA 94089, USA.

MAPPING

The mapping from texture space to screen space is split into two phases. First is the surface parameterization which maps texture space to object space, followed by the standard modeling and viewing transformations which map object space to screen space, typically with a perspective projection [Fol82]. These two mappings are composed to find the overall 2-D texture space to 2-D screen space mapping, and the intermediate 3-D space is often forgotten. This simplification suggests texture mapping's close ties with image warping and geometric distortion.

Scanning Order

There are three general approaches to drawing a texture mapped surface: scanning in screen space, scanning in texture space, and two-pass methods.

Traversing the screen in scanline order, sometimes called inverse mapping, is the most common method. For each pixel in screen space the preimage of the pixel in texture space is found and this area is filtered. This method is preferable when the screen must be written sequentially (e.g. when output is going to a film recorder), the mapping is readily invertible, and the texture is random access.

Traversing the texture in scanline order may seem simpler than scanning the screen since inverting the mapping is unnecessary in this case, but doing this correctly is subtle. Unfortunately, uniform sampling of texture space does not guarantee uniform sampling of screen space except for affine (linear) mappings, so for non-affine mappings texture subdivision must often be done adaptively. Otherwise, holes or overlaps will result in screen space. Scanning the texture is preferable when either (a) the texture to screen mapping is difficult to invert, or (b) the texture image must be read sequentially (e.g. from tape) and will not fit in random access memory.

Two-pass methods decompose a 2-D mapping into two 1-D mappings, the first applied to the rows of an image and the second applied to the columns [Cat80]. These methods work particularly well for affine and perspective mappings, where the warps for each pass are linear or rational linear functions. Because the mapping and filter are 1-D they are amenable to stream processing techniques such as pipelining. Two-pass methods are preferable when the source image cannot be random accessed but it has rapid row column access, and a buffer for the intermediate image is available.

Parameterization

In order to map a 2-D texture onto a surface in 3-D, a parameterization of the surface is needed. This comes naturally for surfaces which are defined parametrically, such as bicubic patches, but less naturally for other surfaces such as polygons and quadrics, which are usually defined implicitly. The parameterization can be by surface coordinates u and v , as in standard texture mapping, by the direction of a normal vector or light ray, as in illumination mapping, or by spatial coordinates x_o , y_o , and z_o for objects which are to appear carved out of a solid material.

Parameterizing Planes and Polygons

We will examine mappings for planar polygons in some detail. First we discuss the parameterization and later we discuss the composite mapping.

A triangle is easily parameterized by specifying the texture space coordinates (u, v) at each of its three vertices. This defines an affine mapping between texture space and 3-D object space; each of x_o , y_o , and z_o have the form $Au + Bv + C$. For polygons with more than three sides, nonlinear functions are needed in general, and one must decide if the flexibility is worth the expense. The alternative is to assume linear parameterizations, and subdivide into triangles where necessary.

One nonlinear parameterization which is sometimes used is the bilinear patch:

$$\begin{bmatrix} x_o & y_o & z_o \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} A & E & I \\ B & F & J \\ C & G & K \\ D & H & L \end{bmatrix}$$

which maps rectangles to planar or nonplanar quadrilaterals [Hou83]. This parameterization has the strange property that it preserves lines and equal spacing along vertical and horizontal texture axes, but preserves neither along diagonals. The use of this parameterization for planar quadrilaterals is not recommended, however, since inverting it requires the solution of

quadratic equations.

A better parameterization for planar quadrilaterals is the 'perspective mapping' [Hec83]:

$$\begin{bmatrix} x_o w_o & y_o w_o & z_o w_o & w_o \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} A & D & G & J \\ B & E & H & K \\ C & F & I & L \end{bmatrix}$$

where w_o is the homogeneous coordinate which is divided through to calculate the true object space coordinates [Rob66], [Fol82]. x_o , y_o , and z_o are thus of the form $(Au + Bv + C)/(Ju + Kv + L)$. The perspective mapping preserves lines at all orientations but sacrifices equal spacing. Note that a 'perspective mapping' might be used for the parameterization whether or not the viewing projection is perspective.

Projecting Polygons

Orthographic Projection

Orthographic projections of linearly-parameterized planar textures have a linear composite mapping. The inverse of this mapping is linear as well, of course. This makes them particularly easy to scan in screen order: the cost is only two adds per pixel, disregarding filtering [Smi80].

It is also possible to perform affine mappings by scanning the texture, producing the screen image in non-scanline order. Most of these methods are quite ingenious.

Braccini and Marino show that by depositing the pixels of a texture scanline along the path of a Bresenham digital line, an image can be rotated or sheared [Bra80]. To fill the holes which sometimes result between adjacent lines, they draw an extra pixel at each kink in the line. This results in some redundancy. They also use Bresenham's algorithm [Bre65] in a totally different way: to scale an image. This is possible because distributing m source pixels to n screen pixels is analogous to drawing a line with slope n/m . Braccini and Marino use the simplest filtering: point sampling.

Weiman also uses Bresenham's algorithm for scaling, but does not draw diagonally across the screen [Wei80]. Instead he decomposes rotation into four scanline operations: xscale, yscale, xshear, and yshear. He does box filtering by averaging together several phases of the scaled image.

Cohen draws texture scanlines diagonally across the screen like [Bra80], but does not use their scaling trick [Coh84]. He is able, however, to eliminate the holes and redundancy of [Bra80] by carefully nesting the digital lines. Cohen also demonstrates the algorithm's applicability to antialiased line drawing.

Perspective Projection

A naive method for texture mapping in perspective is to linearly interpolate the texture coordinates u and v along the sides of the polygon and across each scan line, much as Gouraud or Phong shading [Fol82] is done. However, linear interpolation will never give the proper effect of nonlinear foreshortening [Smi80], it is not rotationally invariant, and the error is obvious in animation. One solution is to subdivide each polygon into many small ones. The correct solution, however, is to replace linear interpolation with the true formula, which requires a division at each pixel. In fact, Gouraud and Phong shading in perspective, which are usually implemented with linear interpolation, share the same problem, but the errors are so slight that they're rarely noticed.

Perspective mapping of an affine or perspective parameterized plane is:

$$\begin{bmatrix} xw & yw & w \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} A & D & G \\ B & E & H \\ C & F & I \end{bmatrix}$$

This mapping is analogous to the more familiar 3-D perspective transformation using 4x4 homogeneous matrices. The inverse of this mapping (calculated using the adjoint matrix) is of the same form, as is the composition of two of these mappings. Consequently a plane using a perspective parameterization which is viewed in perspective will have compound mapping which is of the perspective form. The perspective mapping simplifies to the affine form when G and H are zero, which occurs when the surface is parallel to the projection plane.

Aoki and Levine demonstrate texture mapping polygons in perspective using formulas equivalent to the above [Aok78]. Smith proves that the division is

necessary in general, and shows how u and v can be calculated incrementally from x and y as a polygon is scanned [Smi80]. As discussed earlier, Catmull and Smith decompose perspective mappings into two passes of shears and scales [Cat80]. Gangnet, Perny, and Coueignoux explore an alternate decomposition which rotates screen and texture space so that the perspective occurs along one of the image axes [Gan82]. Heckbert promotes the homogeneous matrix notation for perspective texture mapping and discusses incremental techniques for scanning in screen space [Hec83].

Patches

Texture mapping is quite popular for surfaces modeled from patches, probably for two reasons: (a) the parameterization comes for free, (b) the cost of texture mapping is small relative to the cost of patch rendering. Patches are usually rendered using a subdivision algorithm whereby screen and texture space areas are subdivided in parallel [Cat74], [Lan80]. As an alternative technique Catmull and Smith demonstrate, theoretically at least, that it is possible to perform texture mapping on bilinear, biquadratic, and bicubic patches with two-pass algorithms [Cat80]. Fraser, Schowengerdt, and Briggs explore a similar method for the application of geometric image distortions [Fra85].

FILTERING

After the mapping is computed and the texture warped, the image must be resampled on the screen grid. This process is called *filtering*.

The cheapest texture filtering method is point sampling, wherein the pixel nearest the desired sample point is used. It works relatively well on unscaled images, but for stretched images the texture pixels are visible as large blocks and for shrunk images aliasing can cause distracting moire patterns.

Aliasing

Aliasing can result when a signal has unreproducible high frequencies [Cro77], [Whi81]. In texture mapping, it is most noticeable on high contrast, high frequency textures. Rather than accept the aliasing which results from point sampling or avoid those models which exhibit it, we prefer a high quality, robust image synthesis system which does the extra work required to eliminate it. In practice, total eradication of aliasing is often impractical and we must settle for approximations which merely reduce it to unobjectionable levels.

Two approaches to the aliasing problem are:

- a) Point sample at higher resolution
- b) Low pass filter before sampling

The first method theoretically implies sampling at a resolution determined by the highest frequencies present in the image. Since a surface viewed obliquely can create arbitrarily high frequencies, this resolution can be extremely high. It is therefore desirable to limit dense supersampling to regions of high frequency and high contrast [Cro82] by adapting the sampling rate to the local intensity variance [Lee85], [Dip85]. This is not possible, however, in vectorized algorithms, which must choose a uniform sampling rate a priori and accept any residual aliasing. Whether adaptive or uniform point sampling are used, stochastic sampling can improve the appearance of images significantly by trading off aliasing for noise [Coo86].

The second method, low pass filtering before sampling, is preferable because it addresses the causes of aliasing rather than its symptoms. To eliminate aliasing out signals must be band-limited (contain no frequencies above the Nyquist limit). When a signal is warped and resampled the following steps must theoretically be performed [Smi83]:

1. reconstruct continuous signal from input samples by convolution
2. warp the abscissa of the signal
3. low pass filter the signal using convolution
4. resample the signal at the output sample points

These methods are well understood for linear warps, where the theory of linear systems lends support, but for nonlinear warps such as perspective the theory is lacking and a number of approximate methods have sprung up.

Space Invariant Filtering

For affine image warps the filter is *space invariant*; the filter kernel remains constant as it moves across the image. The four steps above simplify to:

1. low pass filter the input signal using convolution
2. warp the abscissa of the signal

3. resample the signal at the output sample points

Space invariant convolutions are often done using an FFT, multiply, and inverse FFT [Opp75]. The cost of this operation is independent of the kernel size.

Direct Convolution

Nonlinear mappings have space-variant filter kernels (in texture space), which require more complex filtering methods. In general, a square screen pixel which intersects a curved surface has a curvilinear quadrilateral preimage in texture space. Most methods approximate the true mapping by the locally tangent perspective or linear mapping, so that the curvilinear preimage is approximated by a quadrilateral or parallelogram. In place of the ideal low pass filter, a *sinc*, a finite impulse response (FIR) approximation is used to form a weighted average of texture samples.

We now summarize several direct convolution texture filters.

Catmull, 1974

In his subdivision patch renderer, Catmull computes an unweighted average of the texture pixels corresponding to each screen pixel [Cat74]. He gives few details, but it appears his filter is a quadrilateral with a box kernel cross section.

Blinn and Newell, 1976

Blinn and Newell improve on this with a triangular kernel which forms overlapping square pyramids 2 pixels wide in screen space [Bli76]. At each pixel the pyramid is distorted to fit the approximating parallelogram in texture space, and a weighted average is computed.

Feibush, Levoy, and Cook, 1980

The filter used by Feibush, Levoy, and Cook is more elaborate [Fei80].

The following steps are taken at each screen pixel:

- (1) Center the kernel (box, cylinder, cone, or gaussian) on the pixel and find its bounding rectangle.
- (2) Transform the rectangle to texture space, where it is warped into a quadrilateral. The sides of this quadrilateral are assumed to be straight. Find a bounding rectangle for this quadrilateral.
- (3) Map all pixels inside the texture space rectangle to screen space.
- (4) Form a weighted average of the mapped texture pixels using a two-dimensional lookup table indexed by each sample's location within the pixel.

Since the kernel is in lookup table it can be a gaussian or other high quality filter.

Gangnet, Perny, and Coueignoux, 1982

The texture filter proposed by Gangnet, Perny, and Coueignoux is quite similar to [Fei80], but they subdivide uniformly in screen space rather than texture space [Gan82].

Pixels are assumed circular and overlapping. The preimage of a screen circle is a texture ellipse whose major axis corresponds to the direction of greatest compression. A square intermediate supersampling grid which is oriented orthogonally to the screen is constructed. The supersampling rate is determined from the longest diagonal of the parallelogram approximating the texture ellipse. Each of the sample points on the intermediate grid is mapped to texture space and bilinear interpolation is used to reconstruct the texture values at these sample points. The texture values are then weighted by a truncated *sinc* 2 pixels wide in screen space and summed.

The paper contrasts [Fei80]'s "back transforming" method with [Gan82]'s "direct transforming" method, claiming that the latter produces more accurate results because the sampling grid is in screen space rather than texture space. Other differences are more significant. For example, [Gan82] requires a bilinear interpolation for each sample point, while [Fei80] does not. Also, [Gan82] samples at an unnecessarily high frequency along the minor axis of the texture ellipse. For these two reasons, [Fei80] is probably faster than [Gan82] (an estimate denied in [Gan84]).

Greene and Heckbert, 1986

The elliptical weighted average filter (EWA) proposed by Heckbert [Gre86b] is similar to [Gan82] in that it assumes overlapping circular pixels which map to arbitrarily oriented ellipses, and like [Fei80] because the kernel is stored in lookup table, but instead of mapping texture pixels to screen space, the kernel is mapped to texture space, as in [Bli76]. The kernel, a circularly symmetric function in screen space, is warped by an elliptic paraboloid function into an ellipse in texture space. The elliptic paraboloid is computed incrementally and used for both ellipse inclusion testing and kernel table index. The cost per texture pixel is just a few arithmetic operations, in contrast to [Fei80] and [Gan82], which both require mapping each texture pixel from texture space to screen space or vice-versa.

Comparison of Direct Convolution Methods

All five methods have a cost per screen pixel proportional to the number of texture pixels accessed, and this cost is highest for [Fei80] and [Gan82]. Since [Gre86b] has quality comparable to [Fei80] and [Gan82] at much lower cost, it appears to be the fastest algorithm for high quality direct convolution.

Prefiltering the Texture

Even with optimization, the methods above are often extremely slow, since a pixel preimage can be arbitrarily large along silhouettes or at the horizon. We would prefer a texture filter whose cost does not grow proportionately to texture area.

To speed up the process the texture can be prefiltered so that during rendering only a few samples will be accessed for each screen pixel. The access cost of the filter will thus be constant, unlike direct convolution methods. Two data structures can be used for prefiltering: image pyramids and integrated arrays.

Pyramidal data structures are commonly used in image processing and computer vision [Tan75], [Ros84]. Their application to texture mapping was apparently first proposed in Catmull's PhD work [Smi79].

We now summarize several texture filters which employ prefiltering.

Dungan, Stenger, and Suttly, 1978

Dungan, Stenger, and Suttly prefilter their texture "tiles" to form a pyramid whose resolutions are powers of two [Dun78]. To filter an elliptical texture area one of the pyramid levels is selected based on the average diameter of the ellipse and the level is point sampled. The memory cost for this type of texture pyramid is $1 + 1/4 + 1/16 + \dots = 4/3$ times that required for an unfiltered texture; only 33% more expensive.

Smith, 1979

Smith describes the "mipmap", which is a particular layout for color image pyramids invented by Williams [Smi79]. Smith points out that the square filter area inherent in pyramids is inaccurate if the pixel preimage is elongated.

Heckbert, 1983

Heckbert describes Williams' trilinear interpolation scheme for pyramids (see below) and its efficient use in perspective texture mapping of polygons [Hec83]. Choosing the pyramid level is equivalent to approximating a texture quadrilateral with a square. The recommended formula for the diameter d of the square is the maximum of the side lengths of the quadrilateral. Aliasing results if the area filtered is too small, and blurring results if it's too big; one of these two is inevitable.

Williams, 1983

Williams improves upon [Dun78] by proposing a trilinear interpolation scheme for pyramidal images wherein bilinear interpolation is performed on two levels of the pyramid and linear interpolation is performed between them [Wil83]. The output of this filter is thus a continuous function of position (u, v) and diameter d . His filter has a constant cost of 8 pixel accesses and 7 multiplies per screen pixel. Williams uses a box filter to construct the image pyramid, but gaussian filters can also be used [Bur81].

Gangnet and Ghazanfarpour, 1984

In Gangnet and Ghazanfarpour's survey a variation on the image pyramid is proposed which allows unequal filtering in u and v (they call it "asymmetrical" filtering, but is more properly termed "anisotropic"). The image is prefiltered to resolutions of the form $2^{\Delta u} \times 2^{\Delta v}$, so this pyramid is four dimensional: $u, v, \Delta u$ and Δv . Its memory requirements are four times that of an unfiltered image, three times that of an isotropic pyramid, and the time cost is 16 texture pixel accesses and 15 multiplies per screen pixel.

Greene and Heckbert, 1986

Attempting to decouple the data structure from the access function, Greene suggests the use of the EWA filter on an image pyramid [Gre86b]. Unlike the other prefiltering techniques such as trilinear interpolation on a pyramid or the summed area table, EWA allows arbitrarily oriented ellipses to be filtered.

Crow, 1984

Crow proposes the *summed area table*, an alternative to the pyramidal filtering of [Dun78] and [Wil83], which allows orthogonally oriented rectangular areas to be filtered in constant time [Cro84]. The original texture is preintegrated in the u and v directions and stored in a high-precision *summed area table*. To filter a rectangular area the table is sampled in four places (much as one evaluates a definite integral by sampling an indefinite integral). To do this without artifacts requires 16 accesses and 14 multiplies in general, but there is an optimization for large areas which cuts the cost to 4 accesses and 2 multiplies. The high-precision table requires 4 times the memory cost of the original image. The summed area table is generally more costly than the texture pyramid in both memory and time, but it can perform better filtering than the pyramid, since it filters rectangular areas, not just squares. It clearly outperforms the four-dimensional pyramid in [Gan84].

Perlin, 1985

Perlin's *selective image filter* is an elegant generalization of [Cro84], developed independently [Per85b]. If an image is preintegrated in u and v n times, an orthogonally oriented elliptical area can be filtered by sampling the array at $(n+1)^2$ points and weighting them appropriately. The effective kernel is a box convolved with itself n times whose size can be selected at each screen pixel. If $n=0$ the method degenerates to point sampling, if $n=1$ it is equivalent to the summed area table with its box kernel, $n=2$ uses a triangular kernel, and $n=3$ uses a parabolic kernel. With increasing n the kernel approaches a gaussian, and the memory and time costs increase.

Comparison of Prefiltering Methods

The following table summarizes the prefiltering methods we have discussed:

REF.	KERNEL	SHAPE	DOF	TIME	MEMORY
pt samp	impulse	point	2	1,0	1
Dun78	box	square	3	1,0	1.33
Wil83	box	square	3	8,7	1.33
Gan84	box	rectangle	4	16,15	4
Gre86b	any	ellipse	5	?	1.33
Cro84	box	rectangle	4	16,14 or 4,2	4
Per85b	triangle	ellipse	4	36,31 or 9,4	6

The pair of numbers under 'time' is the number of texture pixel accesses and the number of multiplies per screen pixel. The DOF (degrees of freedom) of the filter provides an approximate ranking of filter quality; the more degrees of freedom are available the greater is the kernel shape control.

We see that the integrated array techniques [Cro84] and [Per85b] have rather high memory costs relative to the pyramid methods, but allow rectangular or orthogonally oriented elliptical areas to be filtered. Traditionally pyramid techniques have lower memory cost but allow only squares to be filtered.

Since prefiltering usually entails a setup expense proportional to the square of the texture resolution, its cost is of the same order as direct convolution - if the texture is only used once. But if the texture is used many times, as part of a periodic pattern, or appearing on several objects or in several frames of animation, the setup cost can be amortized over each use.

Filtering in Frequency Space

An alternative to texture space filtering is to transform the texture to frequency space and low pass filter its spectrum. This is most convenient when the texture is represented by a Fourier series rather than a texture array. Norton, Rockwood, and Skolmoski explore this approach for flight simulator applications and propose a simple technique for clamping high frequency terms [Nor82]. Gardner employs 3-D Fourier series as a transparency texture function, with which he generates surprisingly convincing pictures of trees and clouds [Gar85].

Perlin's "Image Synthesizer" uses band limited pseudo-random functions as texture primitives [Per85a]. Creating textures in this way eases transitions from macroscopic to microscopic views of a surface; in the macroscopic range the surface characteristics are built into the scattering statistics of the illumination model, in the intermediate range they are modeled using bump mapping, and in the microscopic range the surface is explicit geometry [Per84]. Each term in the texture series can make the transition independently at a scale appropriate to its frequency range.

Filtering Recommendations

The best filtering algorithm for a given task depends on the texture representation and scanning order in use. When filtering a texture array in a screen order rendering system, the EWA filter [Gre86b], summed area table [Cro84], or selective image filter [Per85b] are recommended because of their good shape control and high speed. Since the above algorithms are still under development and the EWA filter has yet to be tested on a pyramid, it is too early to make definitive judgements. When the texture is a fourier series, filtering is simply a matter of clamping or truncating the high frequency terms [Nor82]. In the case of arbitrary texture functions, which can be much harder to integrate than texture arrays, adaptive stochastic sampling methods are called for [Dip85]. Two-pass algorithms require 1-D space variant texture filters.

Future research on texture filters will continue to improve their quality by providing greater kernel shape control while retaining low time and memory costs. One would like to find a constant-cost prefiltering method which filters arbitrarily oriented elliptical areas using a gaussian kernel.

CONCLUSIONS

System Support for Texture Mapping

So far we have emphasized those tasks common to all types of texture mapping. We now summarize some of the special provisions which a modeling and rendering system must make in order to support different varieties of texture mapping.

The primary requirements of standard texture mapping are texture space coordinates (u, v) for each screen pixel plus the partial derivatives of u and v with respect to screen x and y for good antialiasing (assuming that the rendering program is scanning in screen space).

Bump mapping requires additional information at each pixel: two vectors tangent to the surface pointing in the u and v directions. For facet shaded polygons these tangents are constant across the polygon, but for Phong shaded polygons [Fol82] they vary. In order to ensure artifact-free bump mapping on Phong shaded polygons, these tangents must be continuous across polygon seams. One way to guarantee this is to compute tangents at all polygon vertices during model preparation and interpolate them across the polygon [Max86]. The normal vector can be computed as the cross product of the tangents.

Proper antialiasing of illumination mapping requires some measure of surface curvature in order to calculate the solid angle of sky to filter. This is usually provided in the form of the partials of the normal vector with respect to screen space.

Although they are usually much more compact than brute force 3-D modeling of surface details, texture maps can be bulky, especially when they represent a high resolution image as opposed to a low resolution texture pattern which is replicated numerous times. Keeping several of these in random access memory is often a burden on the rendering program. This problem is especially acute for rendering algorithms which generate the image in scanline order rather than object order, since a given scan line could access hundreds of texture maps. Further work is needed on memory management for texture map access.

General

Texture mapping has become a widely used technique because of its generality and efficiency. It has even made its way into everyday broadcast TV, thanks to new real-time video texture mapping hardware such as the Ampex *ADO* and Quantel *Mirage*. Rendering systems of the near future will allow any conceivable surface parameter to be texture mapped. Despite the recent explosion of diverse applications for texture mapping, a common set of fundamental concepts and algorithms is emerging. We have surveyed a number of these fundamentals: alternative techniques for parameterization, scanning, texture representation, direct convolution and prefiltering.

ACKNOWLEDGEMENTS

I got my introduction to textures while mapping Aspen's facades as part of Walter Bender's *Quick and Dirty Animation System (QADAS)* at the MIT Architecture Machine Group. Pat Hanrahan and Mike Chou contributed enthusiasm and ideas at NYIT's Computer Graphics Lab; Ned Greene provided the cheeseburgers. Thanks also to all the good folks at PDI.

REFERENCES

- Recommended reading: for a good introduction to texture mapping see [Bli76]. [Smi83] gives a helpful theoretical/intuitive introduction to digital image filtering, and [Fei80] goes into texture filtering in detail. The best references on bump mapping and illumination mapping are [Bli78a] and [Mii84], respectively. Systems aspects of texture mapping are discussed in [Coo84b] and [Per85a].
- [Aok78] Masayoshi Aoki, Martin D. Levine, "Computer Generation of Realistic Pictures", *Computers and Graphics*, vol. 3, pp. 149-161, 1978.
 - [Ath78] Peter R. Atherton, Kevin Weiler, Donald P. Greenberg, "Polygon Shadow Generation", *Computer Graphics*, (SIGGRAPH '78 Proceedings), vol. 12, no. 3, Aug. 1978, pp. 275-281.
 - [Bli76] James F. Blinn, Martin E. Newell, "Texture and Reflection in Computer Generated Images", *CACM*, vol. 19, no. 10, Oct. 1976, pp. 542-547.
 - [Bli78a] James F. Blinn, "Simulation of Wrinkled Surfaces", *Computer Graphics*, (SIGGRAPH '78 Proceedings), vol. 12, no. 3, Aug. 1978, pp. 286-292.
 - [Bli78b] James F. Blinn, "Computer Display of Curved Surfaces", PhD thesis, CS Dept., U. of Utah, 1978.
 - [Bra80] Carlo Braccini, Giuseppe Marino, "Fast Geometrical Manipulations of Digital Images", *Computer Graphics and Image Processing*, vol. 13, pp. 127-141, 1980.
 - [Bre65] J. E. Bresenham, "Algorithm for Computer Control of a Digital Plotter", *IBM Systems Journal*, vol. 4, no. 1, July 1965, pp. 25-30.
 - [Bur81] Peter J. Burt, "Fast Filter Transforms for Image Processing", *Computer Graphics and Image Processing*, vol. 16, pp. 20-51, 1981.
 - [Cat74] Edwin E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces*, PhD thesis, Dept. of CS, U. of Utah, Dec. 1974.
 - [Cat80] Ed Catmull, Alvy Ray Smith, "3-D Transformations of Images in Scanline Order", *Computer Graphics*, (SIGGRAPH '80 Proceedings), vol. 14, no. 3, July 1980, pp. 279-285.
 - [Coh84] Ephraim Cohen, "Raster Image Rotation and Anti-Aliased Line Drawing", unpublished, NYIT Computer Graphics Lab, 1984.
 - [Coo84a] Robert L. Cook, Thomas Porter, Loren Carpenter, "Distributed Ray Tracing", *Computer Graphics*, (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 137-145.
 - [Coo84b] Robert L. Cook, "Shade Trees", *Computer Graphics*, (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 223-231.
 - [Coo86] Robert L. Cook, "Antialiasing by Stochastic Sampling", to appear, *ACM Transactions on Graphics*, 1986.

- [Cro77] Franklin C. Crow, "The Aliasing Problem in Computer-Generated Shaded Images", *CACM*, vol. 20, Nov. 1977, pp. 799-805.
- [Cro82] Franklin C. Crow, "Computational Issues in Rendering Anti-Aliased Detail", *Proc. COMPCON '82*, Spring 1982, pp. 238-244.
- [Cro84] Franklin C. Crow, "Summed-Area Tables for Texture Mapping", *Computer Graphics*, (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 207-212.
- [Dip85] Mark A. Z. Dippe, Erling Henry Wold, "Antialiasing Through Stochastic Sampling", *Computer Graphics*, (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 69-78.
- [Dun78] William Dungan, Jr., Anthony Stenger, George Sutt, "Texture Tile Considerations for Raster Graphics", *Computer Graphics*, (SIGGRAPH '78 Proceedings), vol. 12, no. 3, Aug. 1978, pp. 130-134.
- [Fei80] Eliot A. Feibush, Marc Levoy, Robert L. Cook, "Synthetic Texturing Using Digital Filters", *Computer Graphics*, (SIGGRAPH '80 Proceedings), vol. 14, no. 3, July 1980, pp. 294-301.
- [Fol82] James D. Foley, Andries van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.
- [Fra85] Donald Fraser, Robert A. Schowengerdt, Ian Briggs, "Rectification of Multichannel Images in Mass Storage Using Image Transposition", *Computer Vision, Graphics, and Image Processing*, vol. 29, no. 1, Jan. 1985, pp. 23-36.
- [Gan82] Michel Gangnet, Didier Perny, Philippe Coueignoux, "Perspective Mapping of Planar Textures", *Eurographics '82*, 1982, pp. 57-71, (slightly superior to the version which appeared in *Computer Graphics*, Vol. 16, No. 1, May 1982).
- [Gan84] Michel Gangnet, Djarnchid Ghazanfarpour, "Techniques for Perspective Mapping of Planar Textures", *Colloque Image de Biarritz, CESTA*, May 1984, pp. 29-35 (in French).
- [Gar85] Geoffrey Y. Gardner, "Visual Simulation of Clouds", *Computer Graphics*, (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 297-303.
- [Gor84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, Bennett Battail, "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics*, (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 213-222.
- [Gre86a] Ned Greene, "Applications of World Projections", *Graphics Interface '86*, May 1986.
- [Gre86b] Ned Greene, Paul S. Heckbert, "Creating Raster Omnimax Images from Multiple Perspective Views Using The Elliptical Weighted Average Filter", *IEEE Computer Graphics and Applications*, to appear, 1986.
- [Hec83] Paul S. Heckbert, *Texture Mapping Polygons in Perspective*, Technical Memo No. 13, NYIT Computer Graphics Lab, April 1983.
- [Hec84] Paul S. Heckbert, Pat Hanrahan, "Beam Tracing Polygonal Objects", *Computer Graphics*, (SIGGRAPH '84 Proceedings), vol. 18, no. 3, July 1984, pp. 119-127.
- [Hor81] Berthold K. P. Horn, "Hill Shading and the Reflectance Map", *Proc. IEEE*, vol. 69, no. 1, Jan. 1981, pp. 14-47.
- [Hou83] J. C. Hourcade, A. Nicolas, "Inverse Perspective Mapping in Scanline Order onto Non-Planar Quadrilaterals", *Eurographics '83*, 1983, pp. 309-319.
- [Kay79] Douglas S. Kay, Donald P. Greenberg, "Transparency for Computer Synthesized Images", *Computer Graphics*, (SIGGRAPH '79 Proceedings), vol. 13, no. 2, Aug. 1979, pp. 158-164.
- [Lan80] Jeffrey M. Lane, Loren C. Carpenter, Turner Whitted, James F. Blinn, "Scan Line Methods for Displaying Parametrically Defined Surfaces", *CACM*, vol. 23, no. 1, Jan. 1980, pp. 23-34.
- [Lee85] Mark E. Lee, Richard A. Redner, Samuel P. Uselton, "Statistically Optimized Sampling for Distributed Ray Tracing", *Computer Graphics*, (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 61-67.
- [Max86] Nelson Max, personal communication, 1986.
- [Mil84] Gene S. Miller, C. Robert Hoffman, "Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments", *SIGGRAPH '84 Advanced Computer Graphics Animation seminar notes*, July 1984.
- [Nis83] Tomoyuki Nishita, Eiichiro Nakamae, "Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources", *COMPSAC '83, Proc. IEEE 7th Intl. Comp. Soft. and Applications Conf.*, Nov. 1983, pp. 237-242.
- [Nor82] Alan Norton, Alyn P. Rockwood, Philip T. Skolmoski, "Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space", *Computer Graphics*, (SIGGRAPH '82 Proceedings), vol. 16, no. 3, July 1982, pp. 1-8.
- [Opp75] Alan V. Oppenheim, Ronald W. Schaffer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.
- [Pea85] Darwyn R. Peachey, "Solid Texturing of Complex Surfaces", *Computer Graphics*, (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 279-286.
- [Per84] Ken Perlin, "A Unified Texture/Reflectance Model", *SIGGRAPH '84 Advanced Image Synthesis seminar notes*, July 1984.
- [Per85a] Ken Perlin, "An Image Synthesizer", *Computer Graphics*, (SIGGRAPH '85 Proceedings), vol. 19, no. 3, July 1985, pp. 287-296.
- [Per85b] Ken Perlin, "Course Notes", *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*, July 1985.
- [Rob66] Lawrence G. Roberts, *Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs*, MS-1045, Lincoln Lab, Lexington, MA, July 1966.
- [Rog85] David F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, 1985.
- [Ros84] A. Rosenfeld, *Multiresolution Image Processing and Analysis*, Leesberg, VA, July 1982, Springer, Berlin, 1984.
- [Smi79] Alvy Ray Smith, *TEXAS (Preliminary Report)*, Technical Memo 10, NYIT Computer Graphics Lab, July 1979.
- [Smi80] Alvy Ray Smith, "Incremental Rendering of Textures in Perspective", *SIGGRAPH '80 Animation Graphics seminar notes*, July 1980.
- [Smi83] Alvy Ray Smith, "Digital Filtering Tutorial for Computer Graphics", parts 1 and 2, *SIGGRAPH '83 Introduction to Computer Animation seminar notes*, July 1983, pp. 244-261, 262-272.
- [Tan75] S. L. Tanimoto, Theo Pavlidis, "A Hierarchical Data Structure for Picture Processing", *Computer Graphics and Image Processing*, vol. 4, no. 2, June 1975, pp. 104-119.
- [Wei80] Carl F. R. Weiman, "Continuous Anti-Aliased Rotation and Zoom of Raster Images", *Computer Graphics*, (SIGGRAPH '80 Proceedings), vol. 14, no. 3, July 1980, pp. 286-293.
- [Whi81] Turner Whitted, "The Causes of Aliasing in Computer Generated Images", *SIGGRAPH '81 Advanced Image Synthesis seminar notes*, Aug. 1981.
- [Wil83] Lance Williams, "Pyramidal Parametrics", *Computer Graphics*, (SIGGRAPH '83 proceedings), vol. 17, no. 3, July 1983, pp. 1-11.

KEYFRAME-BASED SUBACTORS

L.Forest, D.Rambaud, N.Magnenat-Thalmann, D.Thalmann

MIRALab
HEC/IRO
Université de Montréal
Montréal, Canada

Abstract

The concept of keyframe-based subactor attempts to span two major types of animation: parametric keyframe animation and algorithmic animation. In a keyframe-based subactor, all parameter values are defined by interpolation, however, if there is a law defined for one parameter, this law is applied and values computed by interpolation are ignored. The application of keyframe-based subactors to human motion is also discussed.

keywords: subactor, procedural law, parametric keyframe animation, algorithmic animation

Résumé

On introduit le concept de sous-acteur basé sur des dessins-clés pour tenter de concilier deux types principaux d'animation: l'animation paramétrique à dessins-clés et l'animation algorithmique. Dans un sous-acteur basé sur des dessins-clés, toutes les valeurs de paramètres sont définies par interpolation; cependant si une loi est définie pour un paramètre, cette loi s'applique et les valeurs calculées par interpolation sont ignorées. On décrit aussi une application de ces sous-acteurs basés sur les dessins-clés dans le domaine de l'animation de personnages tridimensionnels.

mots-clés: sous-acteur, loi procédurale, animation paramétrique à dessins-clés, animation algorithmique

Introduction

There have been two major approaches in the design of animation control (Stekettee and Badler 1985; Parke 1982; Zeltzer 1985; Magnenat-Thalmann and Thalmann 1985): keyframe animation and algorithmic animation. The concept of **keyframe-based subactor** attempts to span both types of animation.

Keyframe animation consists of the automatic generation of intermediate frames, called inbetweens, based on a set of keyframes supplied by the animator. There are two fundamental approaches to keyframe animation: **shape-interpolation** and **parametric keyframe animation**.

Shape interpolation is the three-dimensional analog of two-dimensional keyframing, introduced by Burtnyk and Wein (1971). Inbetween frames are computed by interpolating between the data points of the two objects.

In parametric keyframe animation systems (Stekettee and Badler 1985; Parke 1982) inbetween frames are generated by

interpolating the transformation parameters and transforming objects.

In algorithmic animation, motion is algorithmically described. Physical laws are applied to parameters of the objects. Control of these laws may be given by programming as in ASAS (Reynolds 1982) and MIRA (Magnenat-Thalmann and Thalmann 1983) or using an interactive director-oriented approach as in the MIRANIM (Magnenat-Thalmann et al 1985) system. With such an approach, any kind of law may be applied to the parameters. For example, the variation of a joint angle may be controlled by kinematic laws as well as laws based on dynamic analysis (Badler 1984; Armstrong and Green 1985; Wilhelms and Barsky 1985).

In keyframe animation, there are often undesirable effects such as lack of smoothness and discontinuities in motion. To reduce these effects, alternate methods to a linear interpolation have been proposed by Baecker (1969), Burtnyk and Wein (1976), Reeves (1981), Kochanek and Bartels (1984). However, according to Stekettee and Badler (1985), with shape interpolation, there is no totally satisfactory solution to the deviations between the interpolated image and the object being modeled. Unless animators spend their time to digitize almost each frame.

Algorithmic animation is an excellent approach for most motions, however it tends to be complex for specifying human motions. Moreover, kinematic laws may be sometimes completely unrealistic and laws based on dynamic analysis are generally very expensive.

The concept of keyframe-based subactor

An actor as defined by Reynolds (1982) is a graphical entity with a given role to play. A subactor (Magnenat-Thalmann and Thalmann, 1985b) is an entity which is dependent on an actor. This means that all motions applied to an actor are also applied to all its subactors. The reverse is not true. There are also two other advantages to the subactor approach:

1. Any new subactor may be inserted as dependent on an existing actor.
2. Motions of different subactors may be coordinated and synchronized within an actor.

A subactor is a variable of type subactor, which is a data abstraction formulation of a class of entities composed of objects and internal transformations applied to them. Formally a subactor communicates with other entities by means of parameters, which may be time-dependent.

In a keyframe-based subactor, all parameter values are

defined by interpolation, however, if there is a law defined for one parameter, this law is applied and values computed by interpolation are ignored. This approach has great advantages. Most of the parameters may be controlled by the keyframe process, which is less expensive in fact; however more realistic effects may be performed on selected parameters.

Keyframe-based subactors have been implemented as an extension of the MIRANIM system.

Subactors in the MIRANIM system

MIRANIM is an advanced system which allows the creation, manipulation and animation of realistic images. The most important features of MIRANIM are as follows:

- basic geometric primitives
- ruled and free-form surfaces
- multiple cameras and stereoscopy
- actor motions
- multiple lights and spots, shadows (Magnenat-Thalmann and Thalmann, 1985)
- transparency, three-dimensional texture, fractals, particle systems

Image rendering may be performed by a scanline z-buffer algorithm or a ray tracing algorithm.

MIRANIM is mainly based on three components:

- 1) the object modelling and image synthesis system BODY-BUILDING
- 2) the director-oriented animation editor ANIMEDIT
- 3) the actor-based sublanguage CINEMIRA-2

ANIMEDIT is a scripted system; the director designs a scene with decors, actors, cameras and lights. Each of these entities is driven by animated variables, which are, in fact, state variables following evolution laws. CINEMIRA-2 allows the director to use programmers to extend the system. The great advantage of this is that the system is extended in a user-friendly way. This means that the director may immediately use the new possibilities. An entity programmed in CINEMIRA-2 is directly accessible in ANIMEDIT. This not only extends the system, but also enables specific environments to be created. For animation, CINEMIRA-2 allows the programming of five kinds of procedural entities: objects, laws of evolution, actor transformations, subactors, animation blocks.

A CINEMIRA-2 subactor is dependent on an actor which may be transformed in ANIMEDIT by a list of global transformations like translation, rotations, shear, scale, color transformation, flexion, traction. Several actors like these may participate in the same scene with other actors implemented using only algorithmic animation, cameras, lights and decor.

Application of keyframe-based subactors to human motion

A new system has now been designed and implemented: BODY-MOVING; this is a parametric key-frame animation system in which human bodies are mainly controlled by joint angles. BODY-MOVING is an interactive program that allows the user to build any sequence of motion for a given three-dimensional character. Actually, motion is controlled by 50 joint angles. A keyframe is specified by modifying values for these angles from the previous keyframe in the sequence. Corrections may be done vertically for any keyframe, or horizontally for a given parameter in each keyframe. The animator may look at parameter values for any keyframe or

interpolated frame. He/she also may obtain a wire-frame view of the human bodies for any frame.

For each parameter, interpolation may be computed linearly or using bicubic splines (Kochanek and Bartels 1984).

Once the motion of the three-dimensional character is designed, the character needs to be covered with surfaces. In our experimental system, we try to completely separate the topology of the surfaces from the wire-frame model. This means that parts of the human bodies may be designed using ruled surfaces such as revolution surfaces, free-form surfaces or three-dimensional reconstructed surfaces obtained from digitized projections. The system transforms the surfaces according to the wire-frame model assuring an automatic continuity between the different surfaces. This correspondance is based on a changing of reference systems independent of the segment length. This means that for the same set of surfaces, several bodies of different sizes may be obtained according to the segment length in the wire-frame models. This technique may be considered as a three-dimensional skeleton technique (Burtnyk and Wein 1976).

For example consider a point between the elbow and the wrist; when we change the reference system, it is important to notice that both parts may be bent and/or twisted. This means that the surface must be extended on the external side of the elbow and twisted at the wrist, while preserving continuity.

Integration strategy

The integration of parametric keyframe animation and algorithmic animation has been performed considering that any human body designed with BODY-MOVING is a subactor in MIRANIM. This subactor has 50 real parameters, grouped in 17 three-dimensional vector parameters where each component is an angle. Each vector parameter is identified by a name; for example LEFTSHOULDER is the three-dimensional vector that controls the motion of the shoulder. If there is no law defined for this parameter, the interpolated values are taken. If there is a law defined for the parameter, this law is applied and values computed by interpolation are ignored. This approach of keyframe-based subactor has great advantages. Most of the angles may be controlled by the keyframe process, which is less expensive in fact; however more realistic effects may be performed on selected angles. For example, laws based on dynamics have been implemented using similar equations to those described by Armstrong and Green (1985). Of course, to obtain an angle following a law based on the dynamic analysis, dynamic properties like masses, forces, inertia matrices and torques have to be supplied. Intrinsic properties of bodies like masses and moments of inertia may be given at the creation of the surfaces in BODY-BUILDING. Forces and torques have to be specified as parameters of the laws. With our approach, expensive computations are performed only when absolutely necessary. Our approach to the integration of the different techniques is as follows: the joint angles must vary according to the values calculated by BODY-MOVING, but it is also possible to have one or more angles following a predefined law or programmed with the CINEMIRA-2 sublanguage.

To control algorithmically the evolution of an angle, the animator may use three kinds of laws: predefined laws, CINEMIRA-2 analytical laws and functions of a previous state. In this latter case, an evolution law may be completely changed at any time (and consequently at any frame); this allows the animator to adapt the evolution of a joint angle to a new situation.

The integration approach has another important application: the relation between MIRANIM actors and human characters generated by BODY-MOVING. The typical case is when the value of a parameter (angle) of the human character have to be derived from data for an actor. For example, the MIRANIM actor is a ball and the human character receives the ball on the head. In this case, the motion of the character has to be controlled using data about the ball. To solve this case, our approach is to predefine functions which return at any frame the value of any parameter. These laws may be then applied to any animated variable which drives the motion of others actors, cameras or lights.

Fig.1 shows the integration of human keyframe-based subactors into the MIRANIM system.

Acknowledgements

The research was supported by the Natural Sciences and Engineering Council of Canada, the Art Council of Canada and the Government of Quebec.

References

- Armstrong, W.W. and Green, M. (1985) **The Dynamics of Articulated Rigid Bodies for Purposes of Animation**, The Visual Computer, Vol.1, No 4, pp.231-240.
- Badler, N.I. (1984) **Design of a Human Movement Representation Incorporating Dynamics**, Technical Report, Department of Computer and Information Science, University of Pennsylvania.
- Baecker, R. (1969) **Picture-driven Animation**, Proc. AFIPS Spring Joint Computer Conf., Vol.34, pp.273-288.
- Burtnyk, N. and Wein, M. (1971) **Computer-generated Key-frame Animation**, Journal of SMPTE, 80, pp.149-153.
- Burtnyk, N. and Wein, M. (1976) **Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation**, Comm. ACM, Vol.19, No10, pp.564-569.
- Forest, L.; Magnenat-Thalmann, N. and Thalmann, D. (1986) **Integrating Keyframe Animation and Algorithmic Animation of Articulated Bodies**, Proc. Computer Graphics Tokyo '86, Springer, Tokyo.
- Fortin, M.; Léonard, N.; Magnenat-Thalmann, N. and Thalmann, D. (1985) **Animating Lights and Shadows**, Computer-Generated Images, Springer, Tokyo Berlin Heidelberg New York, pp. 45-55.
- Kochanek, D. and Bartels, R. **Interpolating Splines with Local Tension, Continuity and Bias Tension**, Proc. SIGGRAPH '84, pp.33-41.
- Magnenat-Thalmann, N. and Thalmann, D. (1983) **The Use of High-Level Graphical Types in the MIRA Animation System**, IEEE Computer Graphics and Applications, Vol. 3, No 9, pp.9-16.
- Magnenat-Thalmann N. and Thalmann, D. (1985) **Computer Animation: Theory and Practice**, Springer-Verlag, Tokyo.
- Magnenat-Thalmann, N. and Thalmann, D. (1985b) **Subactor Data Types as Hierarchical Procedural Models for Computer Animation**, Proc. EUROGRAPHICS '85, Nice, France, North Holland, pp.121-128.
- Magnenat-Thalmann, N.; Thalmann, D. and Fortin, M. (1985) **MIRANIM: An Extensible Director-Oriented System for the Animation of Realistic Images**, IEEE Computer Graphics and Applications, Vol. 5, No 3, pp. 61-73.
- Parke, F.I. **Parameterized Models for Facial Animation**, IEEE Computer Graphics and Applications, Vol.2, No9, 1982, pp.61-68.
- Reeves, W. **Intbetweening for Computer Animation Utilizing Moving Point Constraints**, Proc. SIGGRAPH '81, Vol.15, No3, 1981, pp.263-269.
- Reynolds, C.W. (1982) **Computer Animation with Scripts and Actors**, Proc. SIGGRAPH'82, pp.289-296.
- Steketee, S.N.; Badler, N.I. **Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control**, Proc. SIGGRAPH '85, pp. 255-262.
- Wilhelms, J.P. and Barsky, B.A. (1985) **Using Dynamic Analysis to Animate Articulated Bodies such as Humans and Robots**, Springer Tokyo Berlin Heidelberg New York, pp.209-229.
- Zeltzer, D. (1985) **Towards an Integrated View of 3D Computer Animation**, The Visual Computer, Springer, Vol.1, No4, pp.249-259.

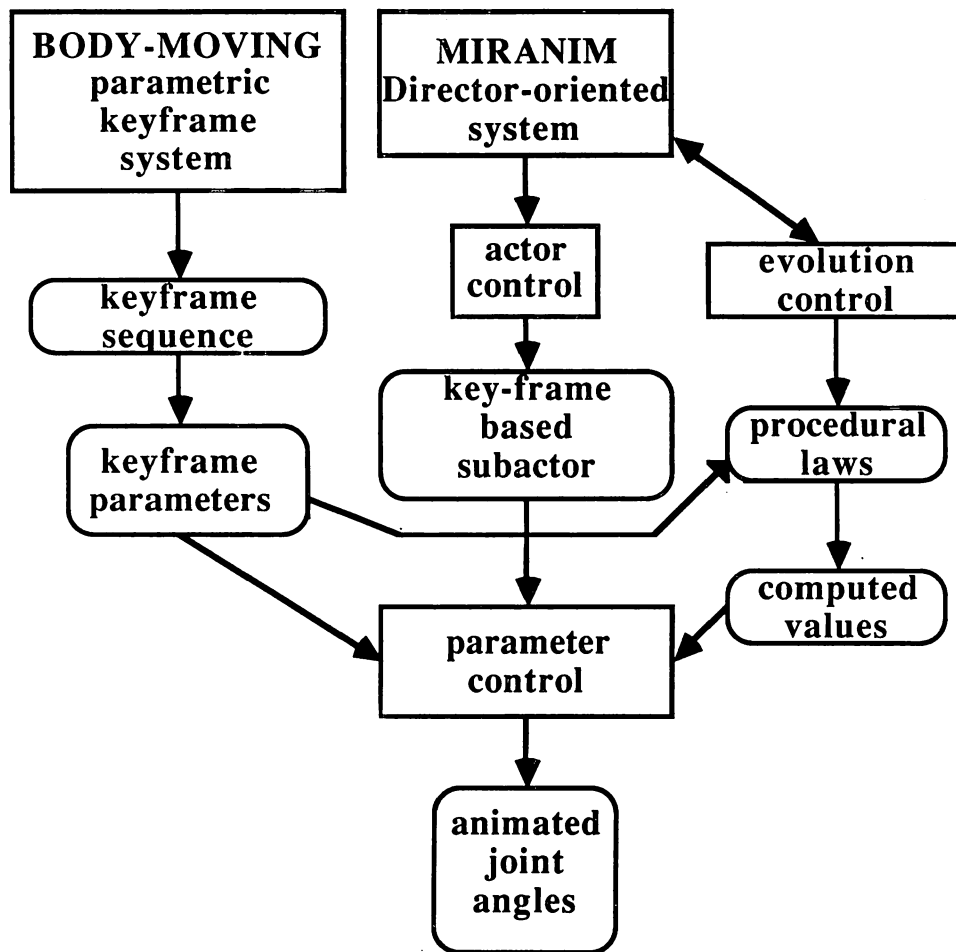


Fig.1 The integration of keyframe-subactors into the MIRANIM system

THE REPRESENTATION OF WATER

Geoff Wyvill

University of Otago Computer Science Department
Box 56, Dunedin, New Zealand

Andrew Pearce and Brian Wyvill

University of Calgary Computer Science Department
2500 University Drive N.W. Calgary, Alberta, Canada

Abstract

Water is the commonest of everyday substances and its many forms have provided subjects for artists for as long as art has existed. But in computer graphics, there seem to have been few attempts to make pictures of water. The reason for this is simple. Realistic pictures of water are very hard to produce. We examine some of the reasons for this difficulty and report on some of our own experiments.

Background

Water is all around us and plays a part in many natural scenes. But it rarely appears in computer graphics images. The last four years has seen an enormous increase in interest in modelling natural phenomena. Fournier [1982] and others have produced realistic terrains using approximations to fractal surfaces building on the ideas of Mandelbrot [1983]. Reeves [1983] has modelled fire and Gardner [1985] has made impressive clouds. Although Reeves suggests that his particle systems can be used to model water, his paper gives no example. Perlin [1985] has published a picture, *Ocean Sunset*, showing a representation of a seascape and this is probably the best representation to date. However, it shows only one appearance of water and it is not clear how the method should be generalised. Water reflections appear in *Road to Point Reyes* [Cook 1983] but they are very simple, as is the pool with ripples in *Erehwon* [Weliky 1985]. Nelson Max [1981] made some pictures of a pair of islands in the sunset and his discussion deals with some of the problems mentioned below. It is, unfortunately, difficult to assess Max's ideas from the pictures themselves as he was hampered by using a display with only 256 colours.

Water is difficult to represent for several reasons. Most of the water we see is in motion. Its shape

depends on this motion and the motion is very complicated. A full, hydrodynamic simulation can, in principle, provide us with a complete description of the shape of any mass of water, but the computational requirements would be huge.

Even when we know the shape of a mass of water, it is still difficult to render because of its optical properties. Most of the light falling on it is refracted or reflected, but even light which passes through the water gets scattered in a more or less complex fashion. And the appearance is further complicated by the fact that any surface below the water is illuminated indirectly by rays focused and scattered by refraction at the surface. Water in lakes, ponds and puddles presents the simplest surface, a plane disturbed by combinations of waves. The wave shapes are affected by varying depth and boundary. Water flowing in streams and rivers is far more complicated. We have initiated a research project aimed at studying all aspects of the appearance of water. In particular we are experimenting with the technique of *soft* objects [Wyvill 1986] to provide a general model for the more complicated cases: streams, waterfalls and fountains.

In this paper, we have confined ourselves to the study of pools of water with waves. We have not yet attempted animation, and we have avoided actual physical simulation. Our main purpose is to discover which features matter most when presenting a realistic picture of water with waves. Another way of looking at it is to ask which features can be omitted without making the picture *unconvincing*. Our standard of assessment is thus rather subjective. Still photographs of moving water often look very different from the original because we never observe waves over an area at the same moment. Ideally we should be trying to compare our artificial pictures with photographs of water in similar circumstances. This we have not done.

This work has been conducted as part of the Graphicsland computer animation project at the University of Calgary [Wyvill 1985]. The Graphicsland system provides a set of programs which are used as tools for creating pictures and animations using three dimensional, computer models. The pictures we have created, all use a ray tracing program which is part of the Graphicsland system. This program has been modified, however, to support some of our special techniques.

Relevant Features

The appearance of water in ponds, pools and puddles varies enormously. Sometimes the water is opaque (or very nearly so) because of mud or other particles and sometimes it is transparent. There is a *range* of colours which are convincing because they do occur in nature whereas other combinations evidently do not. To investigate all this systematically is a huge undertaking but we have made a start by examining those features in earlier work which seem less satisfactory.

Wave Shapes Waves on water are complicated even in the restricted case of pools. The simplest component is a single wave front resulting from some disturbance at one point. The resulting wave consists of an up-and-down motion of water which spreads in all directions from the centre of the disturbance. At any point, the motion can be viewed as growing quickly in amplitude as the wave front first arrives and then slowly decaying. In practice, such a single wave is almost never seen. Wind and other forces continuously make disturbances at many points on the surface and any travelling wave produces secondary, reflected waves wherever it meets the edge of the water.

Nelson Max [1981] went to some trouble to find out what was a reasonable shape for a water wave profile. It seemed to us that the shape produced by just two intersecting waves was so complicated that one would not expect to detect a 'wrong' shape just by eye. Yet Max's sea waves look very wrong in the vicinity of his islands. The reason for this is that the waves do not show any radical change of appearance as they approach the islands. The sloping beach simply intersects what looks like a deep-sea wave. In the presence of an important optical cue such as the beach, we have expectations which must be met if the picture is to look convincing. In *Ocean Sunset* Perlin [1985], we see a remarkably convincing picture of waves on the ocean. Because there are no other objects to give us impressions of scale or expectations about the waves, we are easily convinced. Can we make adequate pictures of water in a context where

we have expectations about the size and behaviour of the waves?

To produce realistic wave shapes on a sloping beach is very difficult so we have chosen to model waves in a rectangular pool with vertical sides. We felt that it was better to use *some* context (unlike Perlin) but we needed to keep it simple enough that we could represent the boundary without recourse to detailed physical simulation.

Max [1981] uses combinations of linear wave fronts. Perlin [1985] observing that these were too regular, combined spherical wave fronts from a collection of point sources whose positions were randomised. Our best results have also been obtained by adding waves from point sources. In some cases we have used a few sources placed at random, in others we have tried to use some knowledge about the scene to choose the source positions. The waves in Figure 5 are generated from a single main source together with eight 'secondary' sources. These secondary sources have been so placed that they represent *reflected images* of the primary source in the sides of the pool. This means that where the waves meet the sides of the pool, their shape is consistent with the expectation that any wave meets a reflection of itself at the edge. Thus the waves of Figure 5 are, in a sense, characteristic of the waves in a rectangular swimming pool.

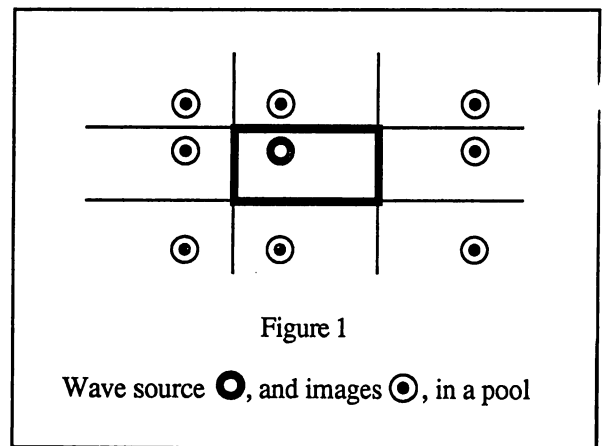


Figure 1

Wave source ●, and images ○, in a pool

Interestingly enough, there is no decay in our waves. They are combinations of sine waves travelling in different directions. The vertical displacement at any point x,y in Figure 5 is:

$$\sum_i w_i \sin(kr_i)$$

Where r_i is the distance from x,y to one of the wave sources and w_i is the amplitude of that source. The reason why this works is not clear. But it is a matter

of common observation that waves in a swimming pool are of pretty uniform size due to the mixing of many disturbances.

Colour Figure 8 shows blue, opaque water. Figure 9 shows green, partially transparent water. Neither is *correct*. What you regard as acceptable depends on your expectations. If the swimmer is in a swimming pool, Figure 8 is more realistic. If he is in a canal, you will probably prefer Figure 9. In no case have we used a very reasonable optical model for the water. The colour in water is a function of depth. Light entering the water is absorbed and scattered so that every part of the water below the surface behaves as a secondary light source and the intensity of these sources is a decreasing function of depth. Light from these sources is further absorbed and scattered, so the colour of a body of water is difficult to predict theoretically even if the shape is very simple and reflection and refraction are ignored. Again, the complexity of this acts in our favour in that we do not know what to expect. In all our pictures, the colour is produced by a simple mixture of light reflected and refracted at the surface. No further filtering or volume dependent effects are used.

Modelling technique Max [1981] uses a functional approximation to the shape of his waves and performs direct ray tracing. The ray intersections are found by iteration. Perlin [1985] is using a scan line algorithm and solid texture mapping. This method is very versatile and enables colour and texture to be changed by post processing the pixel buffer after the scene has been rendered. The method does not, however, support ray tracing so no reflections are possible.

We wanted to compare different approaches so we have used three different techniques. The first is to construct an approximation to our wavy surface using polygons. This is shown in Figure 5. Polygons are not very suitable, but the *Graphicsland* system provides an easy way to create and manipulate such models so this provided us with a convenient reference surface for no extra programming effort.

The second technique is to use our wave function as a texture map. This basically follows Blinn's classical method of bump mapping [Blinn 1978] except that we are ray tracing. The intersection of each ray with the plane water surface is found and then a false surface normal is computed for that intersection point. The direction of reflected and refracted rays is then found using this false normal. This technique is not new. If it has been described explicitly we are not aware of it, but it has certainly been used in published pictures, e.g: *Erehwon* [Weliky 1985]. Bump mapping works

because we cannot perceive the three dimensional position of part of a distant surface very accurately. We deduce the finer detail of shape from light, shade and reflections from the surface. This is another reason why we do not think that the shape of the wave profile is very important. Indeed we do not know what shape our texture-mapped waves are supposed to be, only how the normals are perturbed.

Our third technique is a new application of displacement mapping [Cook 1984]. The idea of a displacement map in scan-line algorithms is to produce local variations of shape by calculating for each surface element a displaced position in space. Using this method, a block can be represented by just a few polygons, and given a curved, sculptured appearance as it is rendered. The advantage of the displacement map is that it enables us to represent the profile where the waves meet the pool side. This feature is missing in the texture-mapped examples.

Experimental Method

We have set up some standard scenes using the tools of *Graphicsland* and rendered them using a specially modified ray tracing program. This has enabled us to generate pictures showing a simulated water surface in which we can change relevant variables selectively. Thus in one picture we make the water opaque, in another transparent. We can use different patterns of surface wave and experiment with our three different rendering techniques.

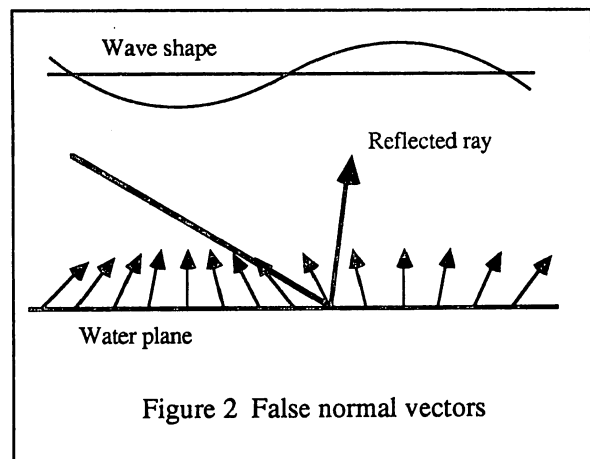


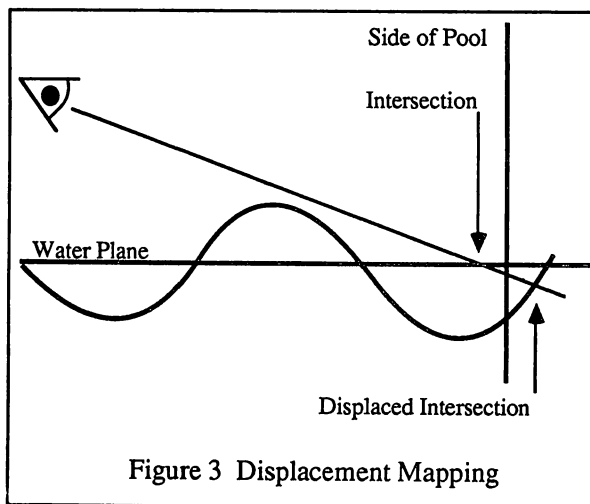
Figure 2 False normal vectors

The Ray Tracer

The principal enhancement of our ray tracer is the ability to recognise ray intersection with an object to which texture mapping applies. Thus the intersection with the water plane is given a surface normal which has been modified by our wave function. This is illustrated in Figure 2. Every point in the plane of the

water surface has its normal vector modified by the wave function. The modifications calculated for all the wave sources are added as vectors to give a normal vector which can deviate from the vertical by a small amount in any direction. This calculation is very fast, but not quite the same as finding the true normal of the combined wave. At this stage, we do not know how important the difference is, but the reflections in Figures 8 and 9 seem good enough.

To produce a profile edge more cheaply than by constructing a genuinely wavy surface, we have also experimented with a displacement map. The idea is that we calculate an ordinary point of intersection with the plane surface which is our water. Then we modify the point of intersection, making it nearer or further from the eye, along the line of sight, according to our wave function. Having got a modified point of intersection we then check back against adjacent objects in case, after all, the point of intersection is now further from the eye than a point of intersection with some other object. For example, in Figure 3, the intersection of the ray and plane surface is in front of the pool's side. But the modified intersection is behind. In this case the ray tracer 'sees' the pool's side, not the water. The effect is to produce an artificial profile edge, Figure 6.



Once again, the wave shape calculated by this displacement map is different from the original. What is worse, the effective shape of the wave depends on the angle of view. Our ray tracer uses a system of uniform space division to reduce the number of ray intersection calculations, and tests for adjacent objects are confined to the current volume of space division. For these reasons the method is less general than we would like, but it does offer a cheap way to represent the wave profile.

Max [1981] reports that in his scenes 10 to 15% of rays were reflected twice by the water surface. Our texture mapping and displacement mapping techniques do not allow for this possibility. We are not sure how important this is. Max shows two pictures rendered with and without this second reflection, but there are other differences in the two scenes which make it difficult to be sure how much this affects our impression of the water.

The photographs

We present a selection of images to illustrate our techniques. Figure 4 is an abstract scene using texture mapping for the water surface. Figure 5 shows the same scene with the polygonal representation. Displacement mapping is used in Figure 6 and the water colour and reflectivity have been changed in Figure 7.

Figures 8 and 9 show the effect of tuning the variables in an attempt at realism. The water in Figure 9 is transparent: note the distorted lower jaw. Our swimmer has no body beneath the head so this picture is a little inconsistent.

Conclusion

We have conducted a series of experiments to determine how to make convincing pictures of water in pools and puddles. We have concentrated on using a simplified model of waves on the surface and a variety of 'tricks' to achieve reasonably fast rendering.

By careful choice of reflectance, colour and other properties, we can make quite convincing pictures of water in this way. But there are many more avenues to be explored. So far, we have made no attempt to simulate the criss cross patterns of light due to refraction and focusing of light sources. This is an important feature of the appearance of shallow water.

Acknowledgement

This research has been partly funded by the Natural Sciences and Engineering Research Council of Canada.

References:

- James F. Blinn 1978
Simulation of Wrinkled Surfaces
Computer Graphics Vol 12 No 3 pp 286-292 August
- Robert L. Cook, Loren Carpenter, Thomas Porter,
William Reeves, David Salesin and Alvy Ray Smith
1983
Road to Point Reyes
Computer Graphics Vol 17 No 3 title page picture,
July
- Robert L. Cook 1984
Shade Trees
Computer Graphics Vol 18 No 3 pp 223-232
- Alain Fournier, Don Fussell and Lorin Carpenter
Computer Rendering of Stochastic Models
CACM Vol 25 No 6 pp 371-384
- Geoffrey Y. Gardner 1985
Visual Simulation of Clouds
Computer Graphics Vol 19 No 3 pp 297-303
- S. Haruyama and Brian Barsky 1984
Using Stochastic Modeling for Texture Generation.
IEEE Computer Graphics and Applications pp 7-19
March
- Benoit Mandelbrot 1983
The Fractal Geometry of Nature
W.H. Freeman and Co
- Ken Perlin 1985
An Image Synthesizer
Computer Graphics Vol 19 No 3 pp 287-296
- William T. Reeves 1983
Particle Systems — A Technique for Modeling a
Class of Fuzzy Objects
Computer Graphics Volume 17 No 3 pp 359-376
- Michael Weliky 1985
Erehwon
Computer Graphics Vol 19 No 3 cover picture, July
- Brian Wyvill, Craig McPheeters and Rick Garbutt
1985
A Practical 3D Computer Animation System
The BKSTS Journal (British Kinematograph Sound
and Television Society), 67 (6) pp 328-332, July
- Geoff Wyvill, Craig McPheeters and Brian Wyvill
1986
Soft Objects
Proceedings of CG Tokyo, to be published

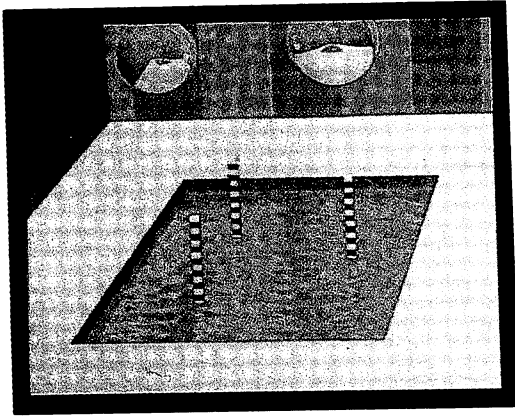


Figure 4 Abstract scene

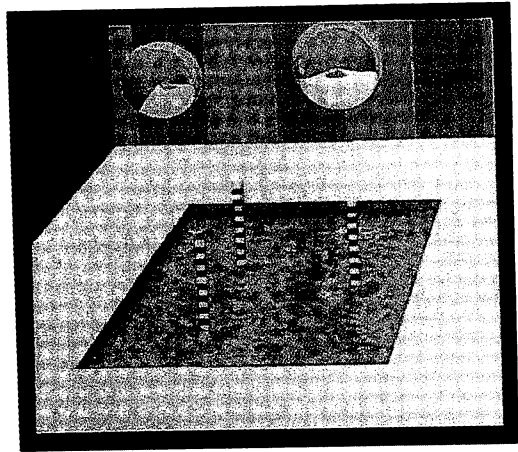


Figure 5 Polygon water

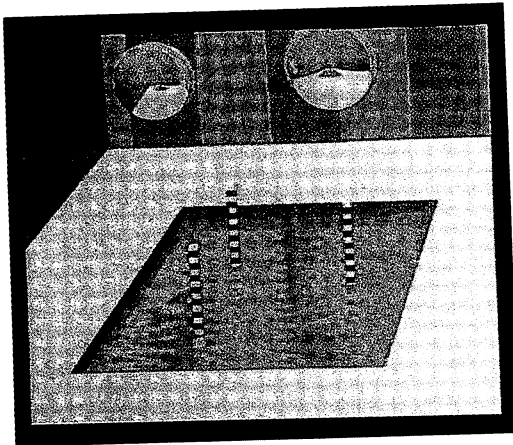


Figure 6 Displacement map

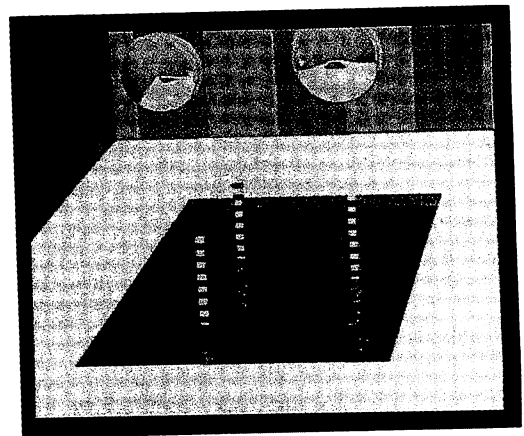


Figure 7 Effect of colour changes

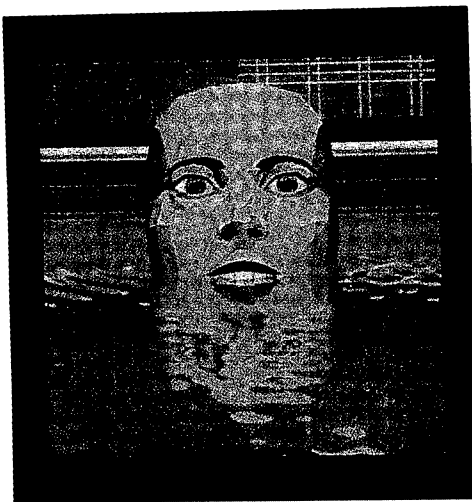


Figure 8 Swimmer

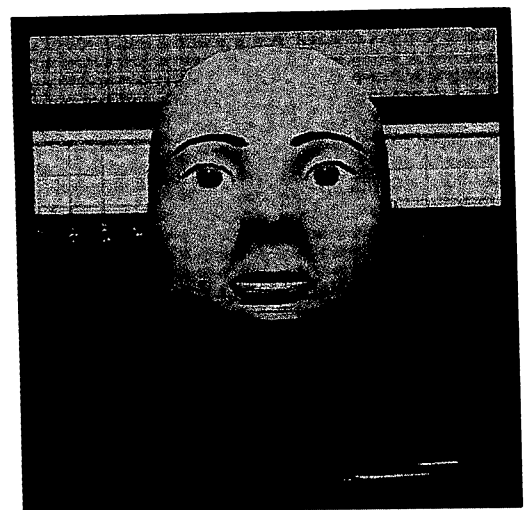


Figure 9 Effect of transparency

PART STRUCTURE FOR 3-D SKETCHING

Alex P. Pentland

Artificial Intelligence Center, SRI International
333 Ravenswood Ave, Menlo Park, California 94025
and

Center for the Study of Language and Information
Stanford University, Stanford California 94038

ABSTRACT

Natural, efficient communication depends upon shared representations. Current 3-D graphics systems, however, use representations that are quite distant from that which people use. The result is that construction of 3-D models is much like programming: meticulous translation from the persons' internal representation to the machines' representation. We argue that a constructive solid geometry representation that allows stereotyped deformations and statistical specification closely parallels peoples' internal representation. Such correspondence allows fast, "natural" 3-D modeling; this is especially important in the initial stages the design process where a "sketching" capability is more important than the ability for precise control of details. We describe and evaluate an interactive system that uses such a representation. The system demands real-time interaction; to support this on 68020-class machines we develop a linear-time hidden line algorithm, so that the hidden-line calculation requires only slightly more time than is needed to draw the lines.

1 Sketching versus Detailing

The distinction between *sketching* and *detailing* is important in understanding how people create a 3-D model. For instance, engineers typically sketch a new part using paper and pencil, and then give the sketch to a draftsman who uses a CAD system to complete the detailed specification of the model. Similarly, animators sketch out scenes and actions before drawing careful renditions of the sequence. The reason that people standardly divide the design process into two stages — each employing its' own media — is that there are two conflicting sets of requirements: the initial design of a 3-D model (i.e., 3-D sketching) demands the ability for quick, general-purpose, and natural interaction, while the final drafting or rendering stage demands the ability for detailed, precise control.

Most current 3-D graphics systems have the wrong "control knobs" for the initial, sketching phase of the design process; that is, the things you would like to do when "roughing in" a 3-D model aren't usually easy to do. This makes things difficult; you have to approach the task of modelling a shape in a planned, methodical manner, much as a programmer approaches the problem of constructing a program¹. Because you have to carefully plan your interaction with the machine, both engineers and graphic artists still sketch shapes on paper before attempting to use a 3-D modeling system.

The use of paper for sketches and computers for final models is bad for exactly the same reasons that the use of paper for final models is bad: lack of flexibility, unneeded duplication of effort, no library of previous drawings, and so forth. In an attempt to address these problems we set out to develop a 3-D modeling language, user interface, and rendering system that is sufficiently "natural" and interactive that people would choose to sketch shapes on the *computer* rather than sketching them out on *paper*.

The idea, then, was to develop a tool that allows the user to very quickly build or modify a 3-D model; to replace the pencil and paper. A user would directly *sketch* 3-D form on the computer, playing with the shape until it looks right, rather than approaching the modeling task as one of entering a carefully predefined model into the computer. An engineer would quickly "sketch" a new part directly on the computer, playing with it until it satisfied him. An animator would "sketch" a scene and, Claymation-like, interactively modify the scene so as to step through key points in an action sequence. In both cases, once we are satisfied with this "sketch model," we can then invest the time to carefully fill out the models' details using a system that is specialized for that particular task.

We want, therefore, a tool that is not specialized to any one application domain but, like pencil and paper, is equally applicable to any 3-D modeling task. And further, like pencil and paper, we want this modeling tool to be generally available: i.e., cheap enough to sit one on everyone's desk, so that they will actually use it.

1.1 The Design of a Graphics System

We have implemented our solution to these problems in a system called SuperSketch (named for "sketching" and "superquadrics"), which provides an environment for interactively sketching and rendering 3-D models. The specific major design criteria for SuperSketch were:

(1) Representation: The system must have a communication metaphor (language) that closely matches the way people naively think about and discuss shape, to promote easy, natural communication between the user and the machine.

(2) Interaction: The system must have an interaction interface that allows users to attain a level of "effortless" interactive control similar to that of an engineer or artist sketching in pencil.

(3) Efficiency and Accessibility: If it is to be truly useful, the system must be efficient enough to allow "real-time" line drawings and rapid full color renderings on a computer inexpensive enough to sit on everyone's desk; e.g., a Motorola 68020-class machine without additional hardware.

In the following sections of this paper we will discuss how we have sought to meet each of these design criteria.

2 Representation

The process of constructing and animating a 3-D model is a process of communication between the machine and the human operator. Because communication depends upon having a shared representation of the situation, the development of natural, "effortless" methods for constructing and animating 3-D shapes depends upon having a representation that is isomorphic to that which people use. When the representation used by the machine doesn't match the way the human operator thinks of

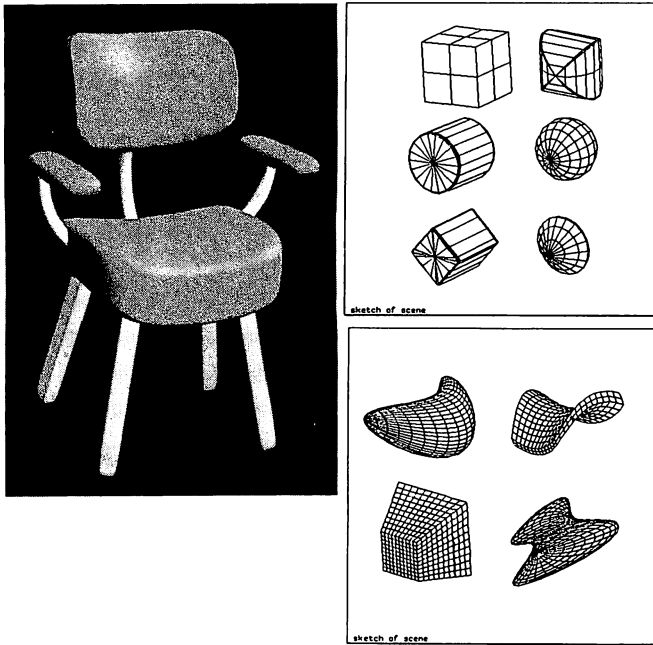


Figure 1. (a) A chair; naive subjects typically describe this as being formed from Boolean combinations of appropriately deformed modeling primitives, (b) a sampling of the basic forms allowed, (c) deformations of these forms.

the process, we get what I call the "Etch-A-Sketch problem²": the system has the wrong control knobs

2.1 Man-machine interaction: building 3-D models

As an illustration of why the way you represent a scene is important, imagine that you were looking at a chair such as is shown in Figure 1(a), and trying to figure out how to build a 3-D model of it. When people verbally describe the shape of this chair, they typically [1,2] say things like

"Well, the back of a chair is a sort of squarish, thin thing that has been bent slightly. The bottom of the chair is the same but thicker, and rotated 90°. The legs are long rectangular things stuck into the bottom of the chair, and ..."

People describe shapes in terms of combining "parts" to form prototypes, and in terms of certain standard deformations of those parts and prototypes. If the computer understood such descriptions, then you could enter the above description of a chair directly. You could construct a 3-D model almost as easily as you could produce a verbal description.

Typically, however, the representation the computer uses is more like splines or polygons, so to enter the model you must adjust spline control points or enter polygons vertices to obtain a shape that matches your mental image of the desired form. Unfortunately, people do not "see" or (normally) think of objects in terms of polygons or splines. Thus the user is forced to carefully (and laboriously) translate between his mental concept of the shape and the computers' representation — to "program" in the base language that the computer uses.

Thus we can liken building a 3-D model on most current day 3-D graphics systems to programming a computer in machine language: you can do anything, but it is often quite laborious. Nor will an elaborate human interface help much: such an interface is like providing the programmer with an assembly language and stepping debugger. Such tools are much better than machine language, but as long as the basic representation is unnatural for the user they still fall short of providing the advantages of a high level language.

Thus it seems that if we could discover a concrete, math-

ematical version of the "parts" that people use to think about 3-D shape, we could construct a graphics system that wouldn't require the user to be a programmer: it wouldn't require him to translate from the way *he* thinks of the problem to the way the *computer* represents the problem.

2.2 Animation

Similar problems arise when we turn from the problem of building 3-D models to the problem of animating them. Polygonal representations, for instance, are too fine grain for ease of manipulation; often the path of each polygon must be separately controlled to produce natural motion. Similarly, spline representations have the problem that non-rigid motions require a very difficult-to-compute interpolation of the spline parameters.

These difficulties arise because the grain size of the representations doesn't match grain size of the problem. Points in the world are not, typically, independent of each other — as they appear in fine-grained polygonal representations — they often move in concert, rigidly or elastically. Larger grain representations such as splines or Constructive Solid Geometry (CSG) systems have the opposite problem, as they assume the relationship between points to be fixed: animators, unfortunately, often want objects to move elastically, and to stretch or compress.

For animation we need to have a representation that matches the grain size of the problem. The disciplines of mechanics, dynamics and kinematics provide a suggestion about how to represent objects for animation, for they represent objects as fixed, solid bodies that undergo translation, rotation and elastic or inelastic deformation.

To model a blade of grass bending in the wind, for example, we would probably first take our polygon or spline description and find a simple mathematical model that was "similar", e.g., a rigid rod. We would then compute the deformation caused by the wind pushing evenly along the length of the rod, and then finally map that deformation back to the polygon or spline representation of the actual shape. It is obvious that things would be simpler if our original representation for the blade of grass were the same one we used for computing the parameters of the bending motion; e.g., a single mathematical object, like the rod, that could then be deformed and rendered directly.

As a more complicated example, consider the modeling of vibrational modes in the animation of biological forms. Muscles, joints and flesh are elastic, and so realistic biological motion must include bouncing and elastic deformation as well as translation and rotation; perhaps the best illustration of this is found in Walt Disney's movies, e.g., the dancing dwarfs in "Snow White and the Seven Dwarfs."

When analyzing the vibrational modes of objects, the standard procedure is to break complex shapes into the union of simple convex shapes whose compression, extension and bending may be separately considered. Thus if we represent our shapes as unions of convex forms with later deformations — similar to the "parts" that people naturally use to describe shape — we will be more easily able to describe, compute, and constrain the parameters of motion because they will be relatively simple functions of the description. That is, a part-by-part description will provide the right "control knobs" for computing the parameters of motion.

In summary, then, the fact that a "part" description is the basis for both peoples' naive notions of form and for mechanics/dynamics/kinematics makes it seem likely that we can develop a descriptive vocabulary that will allow us to accurately model the world in terms of *parts*: a parameterized set of volumetric primitives that, in relatively simple combination, can be used to form rough-and-ready models of the objects in our world and how they behave. If we can develop such part-like modelling primitives then not only will animation become easier, but the problem of building 3-D models will become easier because people seem to think about shape in terms of such part

descriptions. The first question to be answered, therefore, is what is the notion of "a part" that people use?

2.3 People: Parts and Collective Abstractions

A considerable amount is known about how people conceptualize 3-D shape. For instance, we have found that the chair example above is a general phenomenon — i.e., people describe form in terms of combinations of component parts, which in turn are described as modifications of standard prototypes. This sort of structuring of imagery was first explored by the classical Gestalt school of perceptual psychology [3,4], and today is the subject matter of a lively school of investigating human perception [5,6,7]. Indeed, such a part-based, prototype-and-modification descriptive system seems to be common to all human spatial reasoning; the classic work by Rosch [8], for instance, supports this view: she showed that even primitive New Guinea tribesmen (who appear to have no concept of regular geometric shapes) form the geometric prototypes in much the same manner as people from other cultures, and describe novel shapes in terms differences from these prototypes.

Nor is this purely a cognitive phenomenon. When images are stabilized on the retina, for instance, they seem to disappear because low-level mechanisms in the human visual system suppress anything that doesn't move. [This is why you don't see the veins in your retina.] What is interesting is that this disappearance doesn't occur uniformly, but rather affects things in chunks: whole "parts" of objects fade and return, rather than line segments, random patches, or whole objects [9,10].

The central consensus of this research is that people see part boundaries as occurring at places of extremal curvature or at inflections³; this leads to a characterization of 3-D parts as being Boolean combinations (specifically or's and not's) of convex "blobs" [11,12]. When there are specialized cues that indicate that two portions of a figure share a common history — e.g., pronounced axes of symmetry, parallelism, etc. — the human visual system groups these portions together into a single "part" [6,13,14]. Thus we must allow certain stereotyped deformations of our convex blobs to still be considered as a single "part." But which deformations?

We have found that in verbal descriptions of unfamiliar imagery (electron microscope images) people commonly employ a limited set of deformations: bending, tapering, and twisting [1,2,14]. We can also address the question of which deformations are allowable by examining the range of image cues that support the perception of a "deformed part." When we do this we find that the most important grouping cues — symmetry and parallelism — allow reliable inference of bending and tapering, and perhaps of twisting in the case of square-edged or ruled forms [6,13]. Thus we will adopt bending, tapering and twisting as our sole allowable deformations.

Complex natural surfaces. Things seem to happen somewhat differently, however, for complex natural forms such as clouds or mountains, perhaps because such natural shapes simply have too much detail to completely remember, and the details are too variable across instances of the same type of object. Experiments in human memory [15] suggest that for complex surfaces, e.g., a crumpled newspaper, people seem to abstract out a few properties such as "crumpledness" and a few major features of the shape such as the general outline. The rest of the structure is ignored; it is unimportant, *random*.

The fractal-like stochastic representations recently developed in computer graphics mimic this sort of abstraction of qualitative properties like "crumpledness" by letting us qualitatively describe the morass of details by means of a statistical process.

Interestingly, we have found that the parameters of these stochastic processes have a surprising amount of psychological reality. We have shown [16,17], for instance, showing that peoples' perception of "roughness" versus "smoothness" varies as a linear function of the surface's fractal scaling parameter

["fractal dimension"]. This result indicates that representations that incorporate such stochastic models are a start towards duplicating the sort of physically meaningful abstraction of shape that people accomplish.

2.4 A Representational System

The above considerations lead us to the following representational system, a system that we have found competent to accurately describe an extensive variety of natural forms (e.g., people, mountains, clouds, trees), as well as man-made forms, in a succinct and natural manner. The idea behind this representational system is to provide a vocabulary of models and operations that will allow us to model our world as the relatively simple composition of component "parts," retreating to statistical description when the complexity of the scene becomes too large for convenient manipulation.

The most primitive notion in this representation is analogous to a "lump of clay," a modeling primitive that may be deformed and shaped, but which is intended to correspond roughly to our naive perceptual notion of "a part."

For this basic modeling element we use a parameterized family of shapes known as a *superquadrics* [18,19], which are described (adopting the notation $\cos \eta = C_\eta$, $\sin \omega = S_\omega$) by the following equation:

$$X(\eta, \omega) = \begin{pmatrix} C_\eta^{e_1} C_\omega^{e_2} \\ C_\eta^{e_1} S_\omega^{e_2} \\ S_\eta^{e_1} \end{pmatrix}$$

where $X(\eta, \omega)$ is a three-dimensional vector that sweeps out a surface parameterized in latitude η and longitude ω , with the surface's shape controlled by the parameters e_1 and e_2 . This family of functions includes cubes, cylinders, spheres, diamonds and pyramidal shapes as well as the round-edged shapes intermediate between these standard shapes. Some of these shapes are illustrated in Figure 1(b). Superquadrics are, therefore, a superset of the modeling primitives commonly used in CSG systems.

These basic "lumps of clay" (with various symmetries and profiles) are used as prototypes that are then deformed by stretching, bending, twisting or tapering, and then combined using Boolean operations to form new, complex prototypes that may, recursively, again be subjected to deformation and Boolean combination. As an example, the chair in Figure 1(a) was constructed in much the manner that we have found people describe this shape: the back and seats are rounded-edge superquadric "cubes" that are flattened along one axis, and then bent somewhat to accommodate the rounded human form, etc.

The mathematical basis for this portion of the descriptive language was originally developed by Barr [20], although he did not envision it as the basis of a general purpose modeling language. Nonetheless, his work has let us develop a vocabulary of form that closely mimics human notions of part structure and is considerably more powerful than traditional CSG representations.

To illustrate the flexibility of this representation, consider the range of basic superquadric shapes, as shown in Figure 1(b). Already this is a superset of traditional modeling primitives, as it includes rounded shapes as well as traditional Platonic solids. By allowing the deformations that people employ in verbal descriptions — stretching, bending, tapering and twisting — we greatly expand the range of primitives allowed, as shown in Figure 1(c).

Still, the most powerful notion in this language is that of allowing Boolean combination of the primitives. This intuitively attractive CSG approach — building specific object descriptions by applying the logical set operations "or" and "not" to component parts — introduces a language-like generative power that allows the creation of a tremendous variety of form, as is illustrated by the figures in this paper.

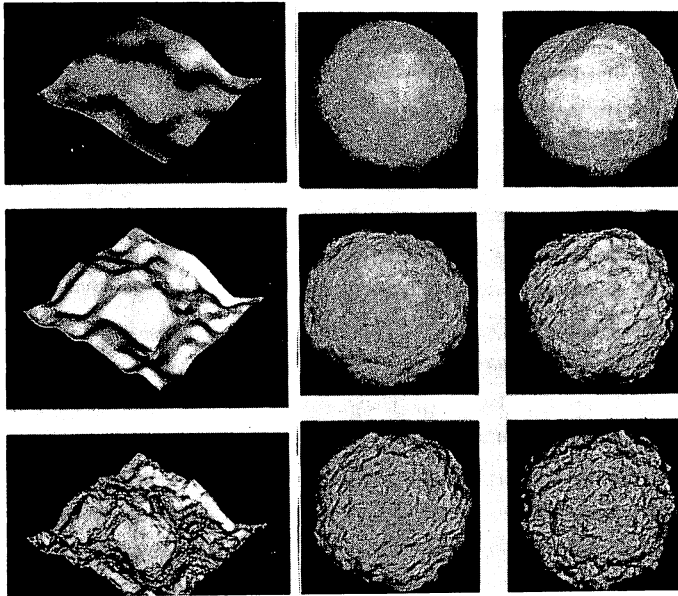


Figure 2. (a) - (c) show the construction of a fractal shape by successive addition of smaller and smaller features with number of features and amplitudes described by the ratio $1/r$. (d) Spherical shapes

2.5 Complex inanimate forms

To show how we may integrate the "part" representation discussed above with the textural abstractions needed to describe complex forms, let us first investigate a model of 3-D texture widely used in the graphics community: fractal Brownian functions. We randomly place n^2 large bumps on a plane (where n is a constant chosen so that the bumps adequately fill out the plane), giving the bumps a Gaussian distribution of altitude (with variance σ^2), as seen in Figure 2(a). We then add to that $4n^2$ bumps of half the size, and altitude variance $\sigma^2 r^2$, as shown in Figure 2(b). We continue with $16n^2$ bumps of one quarter the size, and altitude $\sigma^2 r^4$, then $64n^2$ bumps one eighth size, and altitude $\sigma^2 r^6$ and so forth. The final result, shown in Figure 2(c) is a true Brownian fractal shape. The validity of this construction does not depend on the particular shape of the superquadric primitives employed; the only constraint is that the sum must fill out the Fourier domain.

Different shaped lumps will, however, give different appearance or texture to the resulting fractal surface; this construction, therefore, lets us generalize the standard fractal constructive techniques to produce surfaces with varying lacunarity, etc. One particularly efficient way to produce such shapes is by convolution of appropriately scaled kernels over arrays filled with random noise⁴.

When the placement and size of these superquadric lumps is random, we obtain the classical Brownian fractal surface that has been the subject of much previous research. When the larger components of this sum are matched to a particular object, however, we obtain a description of that object that is exact to the level of detail encompassed by the specified components.

This makes it possible to specify a global shape while retaining a qualitative, statistical description at smaller scales: to describe a complex natural form such as a cloud or mountain, we specify the "lumps" down to the desired level of detail by fixing the larger elements of this sum, and then we specify only the fractal statistics of the smaller lumps thus fixing the qualitative appearance of the surface. Figure 2(d) illustrates an example of such description. The overall shape is that of a sphere; to this specified large-scale shape, smaller lumps were added randomly. The smaller lumps were added with six different choices of r (i.e., six different choices of fractal statistics) resulting in six qualitatively different surfaces — each with the same basic spherical shape.

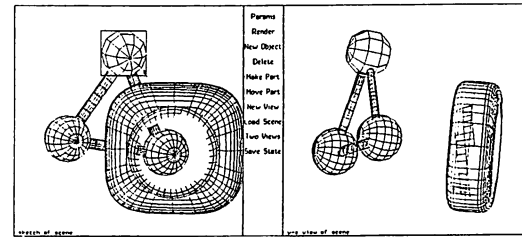


Figure 3. The SuperSketch viewpoints. The left viewport is an interactive view of the scene with hidden lines removed (the linear-time hidden surface algorithm is described in the following section). The right viewport is more like a wireframe model, so that objects are not lost to the users' view.

The ability to fix particular "lumps" within a given shape provides an elegant way to pass from a qualitative model of a surface to a quantitative one — or vice versa. We can refine a general model of the class "a mountain" to produce a model of a particular mountain by fixing the position and size of the largest lumps used to build the surface, while still leaving smaller details only statistically specified. Or we can take a very specific model of a shape, discard the smaller constituent lumps after calculating their statistics, and obtain a model that is less detailed than the original but which is still appears qualitatively correct.

3 Interaction

The first design criterion of our system is a representation (metaphor) that is natural to the human user. The previous section described the metaphor used in this system: that of building objects from clay, a descriptive strategy people often spontaneously use and which they find natural, using our superquadric-based analogy to the human perceptual notion of "parts." Thus the system presents the user with "lumps" of pliable material (like clay) that may then be formed by changing the parameters of the part-like primitives (e.g., modifying the squareness-roundness, length, amount of bending, etc.), and finally combined with other parts of the scene using boolean operations (e.g., "or" and "not").

The second design criterion of our system is that it have a user interface that allows users to attain a level of "effortless" interactive control similar to that of an engineer or artist sketching in pencil.

To provide accurate, complete real-time interactive feedback of the state of the 3-D model under construction, we decided to employ two engineering-style orthographic views (x-y and y-z) of line drawing sketches of the scene. This is shown in Figure 3. All hidden lines are removed in the x-y view (labeled "sketch of scene"), but in the "y-z view" only external facing surfaces are rendered, i.e., objects are seen as "transparent" wireframes, with only back-facing or intersecting portions of the wireframe removed. This partial hidden-surface presentation prevents objects from being lost to the users' view. Objects can be moved, deformed, etc., and redisplayed using a two-hundred triangle line-drawing approximation to the underlying analytical form in about one-eighth of a second, thus providing the perception of smooth, "real-time" motion and deformation.

4 Efficiency and Accessibility

Central to the user's impression of interactivity and "naturalness" is the real-time display of the current state of the 3-D model. Unfortunately, this requirement is in direct conflict with the criterion that our system run on Motorola 68020-class machines.

Polygon-based algorithms are fundamentally order $n \log n$ in the number of polygons, and z-buffer techniques, although linear in the number of polygons, are also linear in the number of pixels. Further, as we require the ability to perform Boolean combinations of our part primitives, we must also add in time for conversion to a standard polygon representation, which is typically order n^2 . Thus achieving real-time display on these

machines seems impossible with current algorithms, because their fundamental computational complexity.

We have therefore developed a hidden line algorithm that is linear in the number of polygons being modified. It is, as far as we have been able to determine, the only example of an incremental, linear-time algorithm other than z-buffer algorithms⁶. This algorithm may be viewed as an analytical version of ray casting [22].

5 Human Interaction Performance

We have set out to build a system that permits a user to quickly sketch a very wide range of form. How well have we really done? There are two ways to answer this question: One, have we developed a representation/metaphor that supports *natural* man-machine interaction?, and two, have constructed a system that permits quick, responsive modeling of form?. Although we have not yet done the sort of careful psychophysical testing that motivated our development of the representation, we can give a subjective evaluation and a few quantitative benchmarks; these are reported below.

A natural vocabulary?. We have found that, as a rule, when we try to model a particular 3-D form using this system we naturally tend to describe the shape in a manner that corresponds to the organization our perceptual apparatus imposes upon the image, even to making the distinctions standardly made in English. That is, the components of the description match one-to-one with our naive perceptual notion of the "parts" in the figure.

For instance, Figure 4 shows how the face is formed from the Boolean sum of several different primitives. The basic form for the head is a slightly tapered ellipsoid. To this basic form is added a somewhat cubical nose, bent pancake-like primitives for ears, bent thin ellipsoids for lips, and almond-shaped eyes, as is shown in Figure 4(a). Figure 4(b) show the addition of rounded cheeks and a slightly pointed chin (is this Yoda from Star Wars?), and finally Figure 4(c) shows the addition of a squarish forehead and slightly fractalized hair.

The smoothly shaded result is shown in Figure 4(d) — it is a reasonably accurate human head, composed of only 19 primitives, specified by slightly less than 130 bytes of information. The two scenes shown in Figure 5 are described in a similarly concise, natural fashion. Figure 5(a) contains only 56 primitives, or about 500 parameters/bytes of information. Figure 5(b) contains only 100 primitives (about 1000 parameters/bytes of information) despite the considerable detailing in the faces (see Figure 4). One should remember that this representation is *not* in any way tailored for describing the human form: it is a general-purpose vocabulary.

The extreme brevity of these descriptions is evidence of their "naturalness." We also note that this brevity makes many otherwise difficult tasks relatively simple, e.g., even NP-complete problems can be easily solved when the size of the problem is small enough. For instance, in animation one would like to be able to specify constraints like "x does not intersect y," "x attached to y," or even "x supports y." When even complex scenes can be described by relatively few "parts" the problem of satisfying constraints can be made tractable.

A quick, responsive system?. The correspondence between the organization of descriptions made in this representation and human perceptual organization means that it is easy to "see" how to assemble a 3-D model. It also means that we try to modify or animate an existing model we will likely find that the changes we have to make are a simple function of the parameters of our model, rather than being, e.g., some hard-to-compute property of a collection of polygons or splines.

Because this part-based representation seems to have the right "control knobs" for manipulating 3-D models, it provides the basis for surprisingly effortless interaction: it took a moderately skilled operator less than a half-hour to assemble the

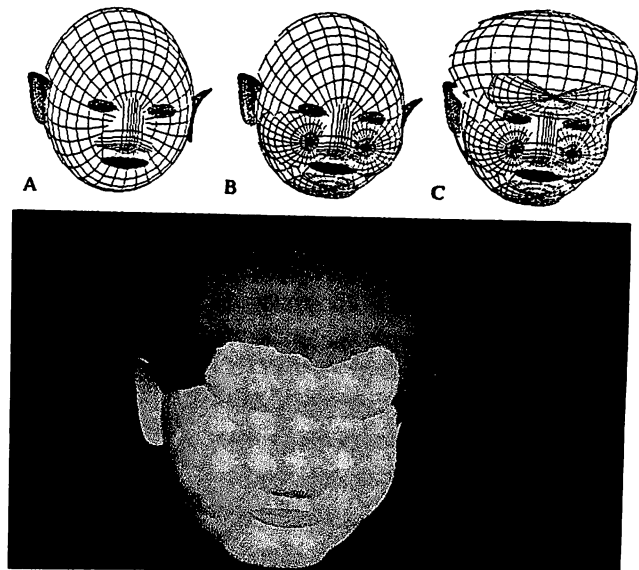


Figure 4. Building a face.

face in Figure 4, about five minutes to create the chair in Figure 1, and less than four hours each (including coffee breaks) to make the images in Figure 5. Much of this speed is due to the brevity of the final descriptions: to build the scene in Figure 5(a), for instance, requires positioning the mouse only 500 times

This performance is in rather stark contrast to more traditional 3-D modeling systems that might require several days to build up a complex scenes such as shown in Figure 5. This performance, perhaps more than any other statistic that could be given, illustrates how the close match between this representational system and the perceptual organization employed by human operators facilitates effective man-machine communication.

6 Summary

Man-machine interaction requires a representation that correctly describes the perceptual organization people impose on the stimulus. We have, therefore, presented a representation that has proven competent to accurately describe an extensive variety of natural forms (e.g., people, mountains, clouds, trees), as well as man-made forms, in a succinct and natural manner. The approach taken in this representational system is to describe scene structure in a manner that is like our naive perceptual notion of "a part," and to allow qualitative description of complex surfaces by means of physically- and psychologically-meaningful statistical abstractions.

To implement this system we have devised a user interface that allows the user to assemble forms in a natural manner, without having to be conscious of the details of either computer or program, and without having to move his hands unnecessarily. This interface requires real-time feedback; to support this we have devised a linear-time hidden line algorithm that allows real-time display of two engineering views of the scene on a 68020-class machine without need for special hardware.

Each of the component parts of this representation — superquadric "lumps," deformations, Boolean combination, and the recursive fractal construction — have been previously suggested as elements of various shape descriptions, usually for other purposes. The contribution of this paper is to bring all of these separate descriptive elements together as a theory of human perceptual organization, and use them as the basis for man-machine interaction. In particular, we believe that the following are the important contributions this paper make toward solving the problems building and animating 3-D forms:

- We have demonstrated that this representational system is able to accurately describe a very wide range of natural and man-made forms in an extremely simple, and therefore useful, manner.
- We have found that descriptions couched in this representation are similar to people's (naive) verbal descriptions and appear to match people's (naive) perceptual notion of "a part."
- We have found that by using the fractal construction with various primitive elements and fractal scaling parameters we can mimic the sort of physically-meaningful statistical abstraction that people seem to employ when describing the shape of complex surfaces.
- And finally, we have shown that descriptions framed in the representation have markedly facilitated man-machine communication about both natural and man-made 3-D structures. It appears, therefore, that this representation gives us the right "control knobs" for discussing and manipulating 3-D forms.

Finally, however, we believe that the representational framework presented here is not complete. It seems clear that additional modeling primitives, such as branching structures [24] or particle systems [25], will be required to model the way people think about objects such as trees, hair, fire, or river rapids. Our future work will involve the integration of these primitives, together with time and motion primitives, into the framework that we have presented here.

REFERENCES

- [1] Teversky, B and Hemenway K., (1984) Objects, parts and categories, *J. Exp. Psychol. Gen.*, 113, 169-193.
- [2] Hobbs, J. (1985) Final Report on Commonsense Summer. SRI Artificial Intelligence Center Technical Note 370.
- [3] Wertheimer, M. (1923) Laws of organization in perceptual forms, in *A Source Book of Gestalt Psychology*, W.D. Ellis (Ed.), New York: Harcourt Brace.
- [4] Johansson, G., (1950) *Configurations in Event Perception*, Stockholm: Almqvist and Wiksell.
- [5] Leyton, M. (1984) Perceptual organization as nested control. *Biological Cybernetics* 51, 141-153.
- [6] Biederman, I., (1985) Human image understanding: recent research and a theory. *Computer Vision, Graphics, and Image Processing*, 32, 29-73
- [7] Hoffman, D., and Richards, W., (1985) Parts of recognition, *Cognition* 18, 65-96
- [8] Rosch, E. (1973) On the internal structure of perceptual and semantic categories. In *Cognitive Development and the Acquisition of Language*. Moore, T.E. (Ed.) New York: Academic Press.
- [9] Prichard, R. M., (1961) Stabilized Images on the Retina, *Scientific American*, 9, 204, 72-79
- [10] Piantinada, T. (1986) Illusory completion under conditions of visual stabilization, to appear, *Science*
- [11] Konderink, Jan J., and van Doorn, Andrea J., (1982) The shape of smooth objects and the way contours end, *Perception*, 11, pp. 129-137
- [12] Marr, D. (1982) *Vision*, San Francisco: W.H. Freeman and Co.
- [13] Binford, T. O., (1971) Visual perception by computer, *Proceeding of the IEEE Conference on Systems and Control*, Miami, December.
- [14] Pentland, A. P., (1986) Perceptual organization and the representation of natural form, *AI Journal* 12, 1-34
- [15] Gregory, R. L., (1970) *The Intelligent Eye*, New York: McGraw-Hill Book Company.
- [16] Pentland, A. (1984a), Fractal-based description of natural scenes, *IEEE Pattern Analysis and Machine Intelligence*, 6, 661-674.
- [17] Pentland, A. (1984d) Perception of three-dimensional textures, *Investigative Ophthalmology and Visual Science*, 25, No. 3, pp. 201.

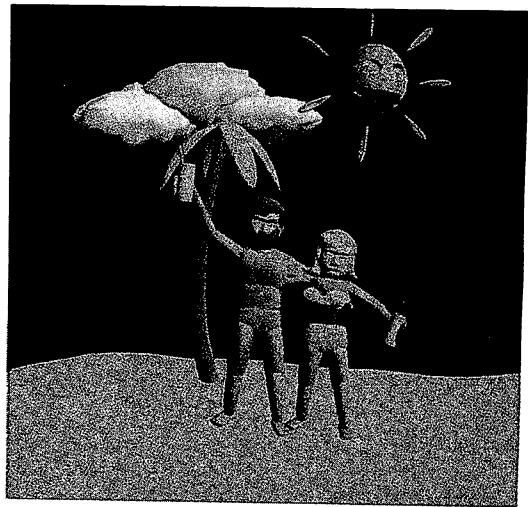
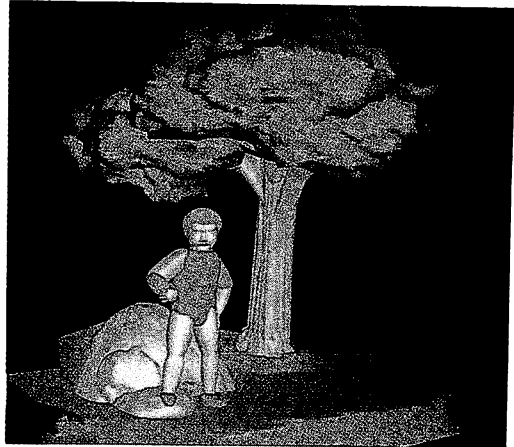


Figure 5. Two scenes that were both constructed in less than four hours, including coffee breaks, on a Symbolics 3600 Lisp Machine (equivalent to a 68020 class machine) without special equipment of any kind. (a) This scene has only 56 primitives (approximately 500 parameters/bytes of information), (b) only 100 primitives are required to model this scene (approximately 1000 parameters/bytes of information), despite the considerable detail in the faces. Both scenes contain about 40,000 polygons; color rendering time is 2.0 minutes.

- [18] Barr, A., (1981) Superquadrics and angle-preserving transformations, *IEEE Computer Graphics and Application*, 1 1-20
- [19] Gardener, M., (1965) The superellipse: a curve that lies between ellipse and rectangle, *Scientific American*, 9, 222-232
- [20] Barr, A., (1984) Global and local deformations of solid primitives. *Computer Graphics* 18, 3, 21-30
- [21] Ogden, J. M., (1985) Generation of fractals using the Burt pyramid. *Optical Society of America Annual Meeting* 14-18 October, p. 39
- [22] Atherton, P. R., (1983) A scan-line hidden surface removal procedure for constructive solid geometry, *Siggraph '83*, 73-82
- [23] Franklin, W. R., A linear time exact hidden surface algorithm *Siggraph '80*, 117-123
- [24] Smith, A. R., (1984) Plants, fractals and formal languages. In *Computer Graphics* 18, No. 3, 1-11.
- [25] Reeves, W. T., (1983) Particle systems - a technique for modeling a class of fuzzy objects, *ACM Transactions on Graphics* 2, 2, 91-108.

Interfacing Image Processing and Computer Graphics Systems
Using an Artificial Visual System

James M. Coggins, Kevin E. Fogarty, and Fredric S. Fay

Biomedical Imaging Group
Department of Physiology
University of Massachusetts Medical School
Worcester, MA
and
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA

ABSTRACT

An Artificial Visual System (AVS) has been developed to simplify three-dimensional microscope images for presentation and manipulation in an interactive computer graphics system. The AVS consists of several sets of spatial filters that decompose an image along three different measurement continua. A recombination algorithm processes the filter outputs to detect objects, to eliminate noise, and to map the detected objects into points in a multidimensional feature space. Recent discoveries regarding the geometry of the points in the feature space are described. One recent result simplifies the AVS by decreasing the number of filters required to obtain the same measurements. Not only are accurate measurements possible, but certain image distortions can be modelled and counteracted in the feature space.

Key words: computer vision, pattern recognition, interactive computer graphics

Introduction

Difficulties in the analysis of natural images arise from random noise, aliasing from the digitization grid, systematic distortions such as optical blur, and from an excess of information -- accurate but irrelevant data, relevant but ambiguous data, or simply too much relevant data -- that we call "information overload". Many techniques exist for correcting noise and distortion [1], but the information overload problem requires an understanding of the aspects of the image that are important for the particular application and techniques for specifying and extracting the relevant aspects of the image. Once the information is extracted, then interactive computer graphics can be applied to present the extracted information and to provide powerful interactive facilities to support image interpretation activity [2].

We have developed an interface between image processing and computer graphics systems that both provides a mechanism for

specifying the salient aspects of the image and extracts that information in a form that can be used to build an interactive graphics model of an image. The interface takes the form of an artificial visual system (AVS).

The development of the AVS as a means for addressing the information overload problem has been motivated by a biomedical research problem involving interpretation of three-dimensional fluorescence images. In this paper we will present an overview of the biomedical research problem and then show how the AVS for this problem was designed. Additional applications of AVS techniques will be discussed.

Background

Our goal is to elucidate the contractile mechanism of smooth muscle cells [3,4]. One of the proteins believed to play a role in that mechanism is α -actinin, which occurs in two types of discrete bodies of concentration: irregular plaques on the cell membrane, and oblong bodies distributed throughout the cytoplasm and oriented within 30 degrees of the long axis of the cell. Organizational patterns such as strands of these bodies branching and twisting through three dimensions may be discerned if the locations of the bodies and the orientations of the oblong bodies are known. Different kinds of organizational patterns could support different hypotheses of cell structure and function.

A three-dimensional image of the protein distribution in a single, isolated cell is obtained by acquiring a series of 2-D optical sections of the cell using Fluorescence Digital Imaging Microscopy [3,5]. Several types of noise are minimized using averaging and normalizing operations during image acquisition [3].

There remains a serious optical distortion in the direction of focus arising from fluorescence sources from out-of-focus planes above and below the focal plane. This distortion has been empirically modelled by imaging a

fluorescent bead smaller than a voxel. The image obtained serves as an empirical estimate of the point spread function of the overall optical system and is used in a constrained iterative restoration procedure to reduce the distortion in the cell images [3]. The restoration reverses about 80% of the optical distortion, but significant distortion remains in the direction of focus. This residual distortion elongates the image of a spherical object in the direction of focus yielding an apparent axial ratio of about 3:1.

The restored image is still difficult to analyze due to information overload. A 64x64x64 cell image can contain over 200 discrete concentration bodies. The oblong bodies are about 1 voxel wide and about 5 voxels long, but their oblique orientation and the residual distortion spread the image of each body over a larger volume. Locating the bodies manually is a tedious task requiring constant flipping between adjacent image planes, correlating traces of the bodies through the planes. Estimating the orientations of the bodies from these traces is even more difficult. In addition to the residual optical distortion, the small apparent size of the bodies makes aliasing from the digitizing grid a serious concern; the difference in the digitized image between two small bodies at nearby orientations involves a subtle shift of energy among a few voxels.

The three-dimensional nature of the data adds more information overload. Since the structures we seek twist through three dimensions, no single two-dimensional view can capture all of the relevant information about the structures.

We need, then, a system to simplify the restored three-dimensional image by locating the protein bodies and determining the orientation of the oblong bodies. The system we have developed is called a three-dimensional artificial visual system.

Designing an Artificial Visual System

An Artificial Visual System (AVS) is a set of filters along some equivalence dimension (e.g. spatial frequency, size, orientation) and a recombination algorithm for mapping the filter responses into a perceptual feature space [4,6]. Defining an AVS involves selecting appropriate equivalence dimensions, designing filters sensitive to different values along those dimensions, and defining the recombination algorithm to perform the visual task at hand.

The equivalence dimension is a continuum along which measurements can be made. Objects in the image will be treated as stimuli to be represented or measured along these continua. Complex structures may require measurements on several equivalence dimensions to adequately characterize the structure of the stimulus.

A sequence of filters is defined for each equivalence dimension. Each filter is sensitive to a different range of values along the continuum. Normally, the sensitivity profiles of the sequence of filters on their equivalence dimension are designed to be tapered and overlapping. Such an ensemble of filters provides a unique sequence of responses for every single-valued stimulus on the equivalence dimension [4,7]. The filter responses serve as coordinates of the stimuli in a multidimensional feature space [8]. Measurements of the stimuli are based on the geometry of the mapping of stimuli into this feature space.

The purposes of the filters are as follows: (1) to decompose the image into separate bands of information so that important and useful information can be identified more easily; (2) to represent the a priori knowledge concerning the objects being sought and the precision of measurement required and (3) to define a feature space into which the image will be mapped [6]. Use of a priori knowledge in the filter design enhances both the sensitivity and the efficiency of the analysis. Sensitivity is enhanced because the filters can be tuned to detect the structures of interest. The filters can also embody the degree of uncertainty in

the a priori knowledge, as demonstrated in the present study where the orientations and sizes of the oblong dense bodies are known a priori only to an approximation. Efficiency is enhanced because the a priori knowledge decreases the number of filters required for the task; a more generalized analysis or higher measurement precision requires a finer or more complete decomposition of the image, requiring more filters. The AVS allows a priori information to be incorporated in the design of the filters rather than in the design of new problem-specific heuristic algorithms.

The recombination algorithm merges the information from each set of filters to perform the visual task required. The algorithm may involve thresholding to eliminate noise, averaging to compute a measurement, location of relative extrema to create a representation or detect a particular kind of stimulus. Other recombination algorithms provide edge detection or texture representations [6,9]. If the objective of the recombination algorithm is to create a representation of the stimulus to be operated upon later by other processes, then the AVS serves as the "low-level vision" component of the vision system. If the objective of the recombination algorithm is to produce the required measurement, then the task is called a "pre-attentive" operation.

The AVS for the Cell Study

For the smooth muscle study, three equivalence dimensions are defined:

length, declination, and azimuth. The filters for each dimension are constructed in the spatial domain and convolved with the 3-D image using Fourier Transform methods. The filtering operation produces three series of filtered images. Fortunately, we can arrange the computations so that the filtered images can be created, processed, and discarded, so it is not necessary to store the entire ensemble of 3-D filtered images at once.

The filters along the length dimension are used to locate the bodies, differentiate them from noise phenomena in the image, and to measure the lengths of the bodies. The filters are shaped (approximately) as truncated cones of length $R=3, 5, 7$, and 9 voxels [4]. See Appendix 2 for details of filter creation. The width of the cone is determined by a priori information concerning the expected variability in the orientations of the oblong bodies.

When convolved with the fluorescence image of the cell, a local relative maximum response occurs when the filter contains a locally maximum fluorescence intensity. These local maxima include the centers of all of the concentration bodies due to the symmetry both of the filters and of the bodies, along with maxima caused by noise or fluorescence hot spots unrelated to the dense bodies we seek.

The recombination algorithm for processing the output of the R series filters involves two steps. First, the local maximum responses are located and thresholded to eliminate miniscule fluorescence hot spots and random noise. Second, if a real concentration body has been found, then the energy captured in the sequence of R filters increases until the filter size exceeds the size of the body, at which point the filter response remains constant. Thus, the length measurement works as follows: If, at a particular location in the image, the smallest filter ($R=3$) has an superthreshold maximum and the $R=5$ filter response at the same location is also a superthreshold maximum that is significantly (1.2 times) greater than the $R=3$ response, then a valid body (length at least 4) has been found. The estimated length will be increased to the size of the next larger R filter as long as the next R filter has a superthreshold maximum at the same location with an intensity greater (1.2 times) than the response of the current R filter.

The orientations of the oblong bodies are measured in terms of the declination ($0 \leq \theta \leq 90$) of the long axis of the body from the y axis of the 3-D image and the

azimuth ($0 \leq \phi < 360$) about the y axis using the x axis as $\phi=0$. (The 3-D images are acquired with the long axis of the cell oriented vertically in the microscope image, corresponding to the y axis of the 3-D image.)

The filters for the theta and phi equivalence dimensions are constructed by partitioning the R filters into "hollow cones" for theta measurements or "wedges" for phi measurements [4]. Theta filters are centered at $\theta=0, 5, 10, 15, 20, 25$, and 30 degrees. Phi filters are centered at $\phi=0, 60, 120, 180, 240$, and 300 degrees. See Appendix 2 for details of the creation of the phi filters. The R filters at $R=5, 7$, and 9 are thus partitioned into theta and phi filters giving a total of 40 filters covering all combinations of values in all three equivalence dimensions.

The 36 theta and phi filters are convolved with the input image. For each body identified by the R filters, we record the responses at the body center of the twelve theta and phi filters corresponding to the size of the body. We use only the theta and phi filters best matching the body size to avoid incorporating responses to nearby structures or noise into the orientation measurements. (There is some evidence that responses to fluorescence signals outside the body can be eliminated mathematically without using separate filter sequences for each possible length. We plan to investigate this possibility later.) These filter responses form two six-dimensional vectors that characterize the theta and phi orientations of the bodies. The feature vectors are normalized by the responses to the R filter matching the body's size so that the sum of the values in each feature vector is 1.

The oblong bodies have a single preferred orientation (by virtue of their oblong shape), so the sequence of responses to overlapping, tapered filters defined along the theta and phi equivalence dimensions is unique for every possible orientation. In fact, the normalized response to each filter may be used as a weighting factor indicating the degree to which the body's spatial energy distribution matches the filter's preferred orientation. The recombination algorithm for constructing an estimate of the orientation of the body involves (circularly) averaging the filter center orientations weighted by the responses of the filters. This is equivalent to a sum of vectors whose polar representation (r, α) has the r component equal to the normalized filter response for the filter centered at $\phi=\alpha$.

Application and Evaluation of the AVS

The performance of the AVS has been extensively studied using artificial images containing model cylindrical bodies at regularly spaced orientations. Model bodies are created at theta angles $0, 5, 10, 15, 20, 25$, and 30 . For each theta angle (except $\theta=0$) bodies are created with phi angles $0, 30, 60, 90, 120, 150, 180, 210, 240, 270, 300$, and 330 degrees. These images have been analyzed themselves, and they have been distorted using the

empirically determined point spread function of the microscope system and partially restored using the iterative restoration algorithm before further analysis.

The objective of this study is to determine from the regular sampling of body angles whether the mapping of the bodies into the feature spaces displays similar regularity. If so, then measurements of body orientations may be reliably performed from the feature space.

With the noiseless images, the R filters correctly locate the bodies, and application of the circular weighted averaging procedure on the sequences of theta and phi filter responses obtains orientation measurements correct within 2 degrees in theta and 5 degrees in phi. The errors remaining are due to aliasing effects and roundoff errors.

With the blurred/restored images, the interpretation of the filtered images is far less straightforward. The residual z-axis distortion, which elongates the images in the $\phi=90$ and $\phi=270$ directions, ruins the theta filter measurements. The problem appears to be that the z-axis distortion causes theta measurements of bodies at a fixed theta orientation to vary through a wide range of values as phi varies. Thus, bodies at different theta orientations are indistinguishable unless the phi angle is already known. We could compute the phi angle first and establish for each phi angle appropriate thresholds for interpreting the theta data, but a more elegant approach has been found.

We have discovered that both the theta and phi measurements can be reliably obtained from the phi filter data alone. This simplification is possible because as the theta angle of a body increases, an increasing proportion of the volume of the body moves away from the $\theta=0$ axis, resulting in increased energy in a preferred phi direction. Therefore, we can measure phi by computing the circular weighted average of the phi filter responses, and we can compute theta by measuring the intensity of the phi angle preference.

When this approach is applied to noiseless model images, the pattern of points in the feature space is a series of concentric circles. Each circle corresponds to a particular theta value, and the polar angle alpha of each point along the circle is precisely the phi angle of the corresponding body.

When the same approach is applied to blurred/restored bodies, the pattern of responses in the feature space is a series of concentric ellipses. The minor axes are in the $\phi=90$ and $\phi=270$ directions, corresponding to the direction of the z axis distortion in the images. The effect

of the distortion on the feature space, then, is to decrease the sensitivity of the filters in the direction of the distortion. We can correct this problem in the feature space by converting the polar (r, α) coordinates to Cartesian form and boosting the y component of the Cartesian vectors. The scaling factor was determined by measuring the eccentricities of the ellipses in the feature space and for our

current data is about 3.2, which corresponds to the axial ratio induced by the residual distortion in the image of a sphere. This scaling factor must be recomputed only if the imaging system changes.

Evaluation of the approach is carried out by measuring the angle between the actual and estimated body orientation vectors. That is, the (θ, ϕ) values of a set of model bodies and the corresponding estimates are converted to Cartesian form and the angle between the two vectors was computed. The average error angle on our model images is less than 2 degrees. Additional tests are in progress on noisy model images. Preliminary results indicate that in the presence of noise, the errors in the orientation measurements increase gradually as the signal/noise ratio decreases. Further work on interpreting these results is in progress.

Interfacing with a Graphics System

The information extracted by the AVS consists of a list of (x, y, z) locations where a dense body was found along with (r, θ, ϕ) measurements on each of the oblong dense bodies. This position and orientation data has been used to create a graphic model of the 3-D distribution of dense bodies. The bodies are represented as lozenge-shaped solid objects having the measured length and orientations. The cell image is created by projecting prototype bodies into space and then subjecting the projected model to the required viewing transformations [2].

The user of the graphics system can specify any view position, including positions inside the graphic model that correspond to positions inside a cell. The three-dimensional distribution of the dense bodies can be explored by marking (in color) particular dense bodies in order to trace a network or follow a strand of bodies through the cell [4].

Interaction is provided by a 3-D wire-frame arrow cursor whose movement is controlled by a three-dimensional joystick. Joystick movements can be interpreted as translation or rotation commands depending on a switch setting.

At present, a single view from a single viewpoint is presented by the graphics system [4]. We are considering implementation of a dual viewport system

that could enable presentation of the view as a stereo pair or as a proximal/distal view pair.

Discussion

The artificial visual system has proven to be a powerful tool for image analysis due to the following properties [4,6]:

1. A priori knowledge can be effectively incorporated into the design of the filters and the recombination algorithm. The artificial visual system can be tuned and experiments can be performed by changing only the filter data and not the analysis algorithms. This enables rapid prototyping and optimization of the artificial visual system with a minimum of reprogramming and algorithm development. In addition, since a few simple algorithms suffice for much of the processing requirements, special devices such as array processors can be brought to bear to enhance the speed of execution.

2. Spatial filtering is intuitively understandable. Filters can be defined either in the spatial domain or in the frequency domain. Either way, the filters and their effects on images can be determined and understood easily since the filter is applied uniformly over the image. Understandability is especially important in applications since decisions will be based on the results of the computer procedures and those decisions must be defended based on an understanding of the computer's results.

3. Fast algorithms exist for performing spatial filtering. Spatial domain convolution, spatial frequency domain multiplication, recursive filtering, and in-place filtering are all well-known algorithms for performing spatial filtering. Depending on the hardware support and the nature of the filtering to be performed, any of these equivalent algorithms can be chosen. These algorithms are amenable to parallel processing to enhance execution speeds.

4. The most important property of spatial filtering is that a suitably constructed ensemble of filters can be used to decompose an image along any of several continua (e.g. size, orientation, spatial frequency, shape, etc.). Thus, the ensemble of filters in a visual system can be constructed so as to define a meaningful feature space.

Moreover, a stimulus can be located along a continuum (orientation, size, spatial frequency) by an ensemble of tapered, overlapping filters. With suitably defined filters, every possible stimulus along the continuum yields a unique pattern of responses from the ensemble of filters, and thus a unique location in the feature space. Increased accuracy in the measurements of stimuli

requires a finer decomposition of the relevant continuum involving a larger number of more narrowly-defined filters. Thus, the tradeoff between cost and measurement accuracy is explicit and measurable.

Conclusion

An Artificial Visual System has been developed to simplify three-dimensional fluorescence microscopy images. The AVS locates bodies of interest in the 3-D image, discriminates the bodies from noise, and measures the 3-D orientation of each body. The measurements are made by using the outputs of a series of spatial filters to map each body into a point in an abstract feature space. The geometry of the mapping allows the orientation angles to be computed directly from the mapping.

Moreover, distortions in the 3-D image due to the image acquisition system that were not corrected by noise reduction or image restoration algorithms appear as systematic distortions of the geometry of the feature space. This residual distortion can be measured and corrected in the feature space, enabling accurate measurements in spite of the imperfections in the image data.

The measurements are then used to create a simplified graphical image that can be viewed and manipulated using interactive graphics tools. Viewpoints corresponding to locations inside a cell may be constructed. The graphics system user can interact with the simplified image to record organizational patterns that may explain the operation of the contractile machinery of the cell.

Appendix 1: Coordinate Conversions

This appendix gives the algorithms for converting Cartesian vectors to (theta,phi) orientation vectors and vice versa where the (0,0) direction is the y axis and the (90,0) direction is the x axis.

Convert a (theta,phi) orientation to a 3-D Cartesian unit vector [x,y,z] as follows:

```
x = sin(theta)*cos(phi)
y = cos(theta)
z = sin(theta)*sin(phi)
```

The following algorithm converts a 3-D Cartesian vector [x,y,z] to a (theta,phi) orientation vector:

```
Let r1 = sqrt(y^2+z^2)
theta = arctan(r1/y) if y is not zero
      = 0           if y=0 and r1=0
      = 90          if y=0 and r1>= 0
phi    = arctan(z/x) if x is not 0
      = 0           if x=0 and z=0
      = 90          if x=0 and z>=0
```

Appendix 2: Filter Definitions

Filters defined as geometric cones and segments of cones [4] were found to be subject to errors such as false positives or mislocated maxima. Superior results have been obtained with filters created by integrating images of a cylinder rotated about its center. The resulting filters have higher sensitivity where many of the rotated cylinder images overlap (i.e. at the filter center).

Each filter is formed by a weighted sum of images of a rotated cylinder 9 voxels long and 1 voxel in diameter corresponding to the longest apparent size of the dense bodies in our images. The weighting factor applied to each cylinder is based on the difference between the cylinder orientation (t,p) and the filter's preferred orientation. We define two utility functions as follows:

$$WT(t,p;tmax) = 1 \quad \text{if } t \leq tmax \\ = \max(0, 1 - [(t - tmax)/10]) \quad \text{if } t > tmax$$

$$WP(t,p;pcen) = \max(0, 1 - [|pcen - p|/60])$$

The WT function assigns a weight of 1 to cylinders whose theta orientation is less than or equal to tmax and attenuates cylinders with larger theta values with the weight decreasing linearly with (t-tmax) and reaching 0 at a theta orientation of tmax+10 degrees. The WP function attenuates the cylinder images linearly as the phi orientation differs from pcen with the weight reaching zero 60 degrees away from pcen. Note that the difference (pcen-p) must be computed mod 360. Now we use the utility functions to define the cylinder weighting functions for an R filter (a cone) and filters P1-P6 (wedge-shaped phi filters):

$$\begin{aligned} R(t,p) &= WT(t,p;30) \\ P1(t,p) &= WT(t,p;30) * WP(t,p;0) \\ P2(t,p) &= WT(t,p;30) * WP(t,p;60) \\ P3(t,p) &= WT(t,p;30) * WP(t,p;120) \\ P4(t,p) &= WT(t,p;30) * WP(t,p;180) \\ P5(t,p) &= WT(t,p;30) * WP(t,p;240) \\ P6(t,p) &= WT(t,p;30) * WP(t,p;300) \end{aligned}$$

The R filter is equal to the sum of the Pi filters, making the response of the R filter a reasonable normalization factor for the sequence of Pi responses. This normalization eliminates the effects of different overall intensity in different bodies.

Shorter filters are created by multiplying the above filters by a sphere of an appropriate radius. R and Pi filters have been created at lengths 3, 5, 7, as well as 9. When the size of a body is determined, the P series filters corresponding to that size are used to estimate the theta and phi angles.

References

1. A. Rosenfeld and A. C. Kak, Digital Picture Processing, second edition, New York: Academic Press, 1981.
2. J. D. Foley and A. vanDam, Fundamentals of Interactive Computer Graphics, Reading, MA: Addison Wesley, 1984.
3. F. S. Fay, K. E. Fogarty, and J. M. Coggins, "Analysis of Molecular Distributions in Single Cells Using a Digital Imaging Microscope," in Optical Methods in Cell Physiology, P. de Weer and B. Salzberg, editors, John Wiley and Sons, New York, 1985.
4. J. M. Coggins, K. E. Fogarty, and F. S. Fay, "Development and Application of a Three-Dimensional Artificial Visual System," Proc. Symposium on Computer Applications in Medical Care, Nov. 10-13, 1985, Baltimore, MD. pp. 686-690. Also in press in Journal of Computer Methods and Programs in Biomedicine.
5. D. J. Arndt-Jovin, M. Robert-Nicoud, S. J. Kaufman, and T. M. Jovin, "Fluorescence Digital Imaging Microscopy in Cell Biology," Science, vol. 230, no. 4723, 18 Oct 1985, pp. 247-256.
6. J. M. Coggins, A Framework for Texture Analysis Based on Spatial Filtering, Ann Arbor: University Microfilms, dissertation for Ph.D. Michigan State University, 1982.
7. W. Richards, "Quantifying Sensory Channels: Generalizing Colorimetry to Orientation and Texture, Touch, and Tones," Sensory Processes v. 3, 1980.
8. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, New York: Wiley.
9. J. M. Coggins and A. K. Jain, "A Spatial Filtering Approach to Texture Analysis," Pattern Recognition Letters, v. 3, 1985, pp. 195-203.

CONNECTED COMPONENT LABELING USING MODIFIED LINEAR QUADTREES

Xiaoning Wang and Wayne A. Davis

Department of Computing Science
The University of Alberta
Edmonton, Alberta, Canada T6G 2H1

Abstract¹

Using the modified linear quadtree proposed in [1,9], this paper presents an $O(n \cdot N)$ algorithm for labeling connected components of a region consisting of N BLACK nodes in a 2^n by 2^n binary image. As a direct application of the algorithm, a method for computing the perimeter of a region is also described.

1. INTRODUCTION

The identification of all connected components of a region is a fundamental operation in image processing and geographic systems [4, 5]. Samet [6] presents an algorithm for labeling all connected components of a region represented by a quadtree, and shows that its average execution time is $O(T + N \cdot \log N)$, where T and N are the total number of nodes and the number of BLACK nodes in the quadtree, respectively. That algorithm outperforms the traditional method which has an execution time proportional to the number of pixels of the image [5]. Gargantini [3] also describes an entirely different algorithm using a linear quadtree [2]; however, that algorithm has limited power as it is only applicable to regions with very special configurations.

In this paper, an algorithm adopting a novel approach for labeling all connected components of a region using a Modified Linear Quadtree (MLQ) is presented. It is capable of handling regions with arbitrary configurations. Furthermore, the algorithm is of time complexity $O(n \cdot N)$, and hence compares favorably to Samet's algorithm [6]. As an application of the algorithm, this paper will show that, with the same time complexity, the perimeter of a region can also be computed.

2. DEFINITIONS AND NOTATION

This section contains some basic definitions and terminology for region representations that are fundamental for the remainder of this paper.

¹This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant NSERC A7634.

Definition 1: An *image* is a 2^n by 2^n array of unit square pixels each of which can assume one of 2^k values, where n is called the *resolution parameter* of the image.

Definition 2: An image is called a *binary image* when its pixels assume either 1 or 0 values. A pixel is BLACK if it has the value of 1, otherwise it is WHITE.

Without loss of generality, only binary images will be considered in this paper since all of the algorithms can be extended to nonbinary images.

Definition 3: The *region* of a binary image is composed of all BLACK pixels, and the *background* of the region is composed of all WHITE pixels.

Definition 4: Let (i, j) represent the location of a pixel p in a given image, where i and j are the column and row positions respectively. Then p has four horizontal and vertical neighbors located at: $(i-1, j)$, $(i, j-1)$, $(i, j+1)$ and $(i+1, j)$. These pixels are called the *4-neighbors* of p , and are said to be *4-adjacent* to p .

Definition 5: For two BLACK pixels, p and q , of a region, p is said to be *connected* to q if there is a path from p to q consisting entirely of pixels of the region.

Definition 6: For any BLACK pixel p , the set of pixels connected to p is called a *connected component* of the region. If a region has only one component, then it is called "connected".

Based on the principle of recursive decomposition, an image is decomposed in the following manner to separate a region from its background[10]. If the region does not cover the entire binary array, the array will be subdivided into four equal-sized square blocks. This process will be applied recursively, until blocks are obtained that are either totally contained in the region or totally disjoint from it. The recursive decomposition of an image produces blocks that must have standard sizes (powers of 2) and positions. Similar definitions can now be formulated in terms of blocks.

Definition 7: A block is said to be BLACK if it contains only BLACK pixels, WHITE if it contains only WHITE pixels, and GREY if it contains both BLACK and WHITE pixels.

The four sides of a block are referred as to its North, East, South and West sides, or N, E, S and W for short. Let $OPSIDE(T)$ be the side opposite to T, e.g., $OPSIDE(E)=W$.

Definition 8: Two blocks P and Q are said to be 4-adjacent along the side T of P if the side T of P touches the side $OPSIDE(T)$ of Q.

Definition 9: BLACK blocks P and Q are said to be connected if there exists a path consisting entirely of BLACK pixels from a pixel of P to a pixel of Q.

Definition 10: For two integers I and J given by

$$I = \sum_{i=0}^{n-1} (I_i \cdot 2^i), \text{ and } J = \sum_{i=0}^{n-1} (J_i \cdot 2^i), \text{ where } I_i, J_i \in \{0,1\},$$

$$SHUFFLE(I,J) = \sum_{i=0}^{n-1} (I_i \cdot 2 + J_i) \cdot 4^i.$$

To represent a block obtained by the recursive decomposition method requires the following definition:

Definition 11: The key of a block or node with 2^s by 2^s pixels is $SHUFFLE(I, J)$, where (I, J) is the location of its left bottom pixel, and s is the resolution parameter of the block.

It is now easy to show that the two-tuple $\langle K, s \rangle$ uniquely represents a block, where K and s are the key and resolution parameter of the block, respectively.

A modified linear quadtree (MLQ) is defined to be a sequence of BLACK nodes in two-tuple form sorted in ascending key order. This differs from the usual definition of a linear quadtree in that the key of the node is stored as a single integer rather than as an n-digit quaternary code, and the resolution parameter of the node is given explicitly rather than implied by the number of don't care characters in the quaternary code. This modification results in space efficiency and improved execution time [9].

In presenting the connected component labeling algorithm, each BLACK node in the MLQ is stored as a record consisting of three fields. The first two fields, termed KEY and RES, contain the key and the resolution parameter of the node, respectively. The third field, termed ID, identifies the connected component containing the node. It is set as a result of the algorithm to be presented. An array M is used to represent the MLQ. Therefore, M has the property that for any

$i, j = (1, 2, \dots, N)$, if $i < j$ then $M[i] \cdot KEY < M[j] \cdot KEY$.

The predicate $UNEXPLORED(P, T)$ is true if and only if the side T of node P has not been marked "explored" in the progress of the algorithm. The predicate $LABEL(P)$ is true if and only if P.ID has been assigned a value.

3. AN OBSERVATION

Given a node P in M, its four adjacent or neighboring nodes can be determined in $O(n)$ steps [1,9]. Suppose Q is the adjacent node to P in the west direction. The color of Q can be determined as WHITE, BLACK or GREY in $O(\log N)$ time [1,9]. The BLACK or WHITE color of Q provides the information regarding whether Q is connected to P or not. Very little knowledge, however, of what is happening between P and Q is known when the color of Q is GREY. Simply, this is because there can either be no BLACK node or as many as up to 2^s BLACK nodes in Q adjacent to P, where $s = P.RES$, i.e., P is a block of 2^s by 2^s pixels.

This implies that up to 2^s further searches on M must occur in order to exhaust all possible adjacencies. In fact, this is precisely what Samet's algorithm does. Assuming a random image, in the sense that a node is equally likely to appear in any position and at any level in the quadtree, the neighbor finding operation using a quadtree is so efficient that the average number of nodes visited is a constant [8]. Correspondingly, the neighbor finding operation using a linear quadtree is less efficient in that the average number of nodes visited is $O(\log N)$ [2]. Therefore, a connected component labeling algorithm using a linear quadtree cannot do the same thing as Samet's algorithm does.

Gargantini's algorithm [3] imposes a special configuration on the region to avoid performing an exhaustive search. As a result, the algorithm is not able to deal with regions with arbitrary configurations. Clearly it is a crucial step, in achieving an efficient method that when Q, the adjacent node of P, turns to be GREY, of how to preclude further searching on M without losing any information regarding the adjacencies.

It is this observation that leads to a new method, to be described in the next section, for labeling all connected components of a region using an MLQ.

4. AN INFORMAL DESCRIPTION

The connected component labeling algorithm has three phases. An array called MAP will be used mainly by the first phase. MAP is constructed from M such that for any two integers, $i, j = (1, 2, \dots, N)$, if $i < j$ then $M[MAP[i]] \cdot RES \leq M[MAP[j]] \cdot RES$. In essence, the use of MAP provides the visit of the nodes in M in ascending size order, while traversing M.

The first phase explores all possible adjacencies between any pair of BLACK nodes in M and generates

equivalence pairs. The second phase merges all the equivalence pairs generated during phase one into equivalence classes. Finally, the third phase assigns the same identifier (i.e., the label) to those BLACK nodes that belong to the same equivalence class to reflect a connected component.

In particular, phase one traverses M in ascending size order. For each BLACK node P in M being visited, and T in $\{N, E, S, W\}$, if the side T of P has not been previously marked, then the adjacency between node P and the BLACK node of greater or equal size along the side T of P needs be explored. If such a BLACK node indeed exists in M , say Q , then the side **OPSIDE**(T) of Q is marked "explored" and is assigned the same label as that of P to indicate that both P and Q belong to the same component. Depending on the configuration of the region under consideration, Q may already have been assigned a label different from that of P , in which case, an equivalence pair consisting of the two labels is generated. This equivalence pair will be used in the later stages of the algorithm to update the labels of P and Q so that eventually they will be assigned the same label. If the side T of P has already been marked "explored", then the exploration of the adjacency to the side T of P is no longer needed.

The consequence of this technique is not only to save one search on M , but rather to save the necessity of exhausting all possible adjacencies along the side T of P . The reason for this is as follows. The side T of P can be marked "explored" only at the time when that side of P was found to be connected to a BLACK node that was being visited by the algorithm. The size of this BLACK node cannot be bigger than P for otherwise it would not be visited before P . As a matter of fact, there could be as many such BLACK nodes as the size of P in M . Regardless how many BLACK nodes of this nature exist, the "explored" status of the side T of P , while P is being visited, simply indicates that the exploration of the adjacencies across the side T has been previously done.

The distinct feature of this algorithm is that phase one guarantees that, at most, one exploration of an adjacency along each side of every BLACK node in M is sufficient to discover all possible adjacencies between any pair of BLACK nodes. To see this, remember that phase one visits the nodes in M in ascending size order. Consider, for example, the image in Fig. 1, where the resolution parameter n is 3. By the time BLACK node A is visited, its eastern adjacency needs not be re-explored, since BLACK nodes E , D , C and B have already been visited before A , and the adjacencies were discovered at that time. Now, however, its northern adjacency must be explored, since that side of A cannot be marked "explored" although F was visited before A . As A 's northern neighbor of equal size is found to be GREY,

the algorithm immediately concludes that there does not exist a BLACK node adjacent to the northern side of A , for otherwise the northern side of A would have been marked "explored". Therefore, no further search is necessary.

Phase two will merge the equivalence pairs generated during phase one into equivalence classes in such a way that each equivalence class contains all labels assigned to those BLACK nodes that form a connected component.

Finally, phase three updates the labels assigned to the BLACK nodes during phase one using the equivalence classes generated by phase two. Upon completion of phase three, all BLACK nodes of each connected component will have unique labels.

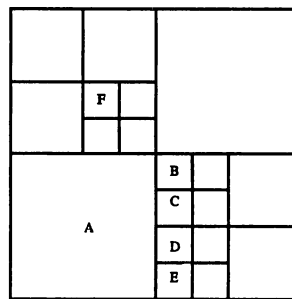


Fig. 1. An Adjacency Configuration.

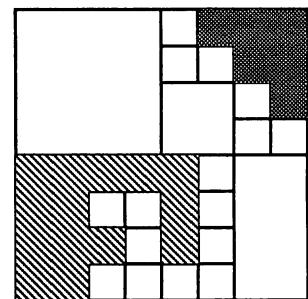


Fig. 2. A Region With 2 Components.

5. THE FORMAL ALGORITHM

The connected component labeling algorithm will now be specified by the following procedures. Actually, only those procedures that correspond to phases one and three will be presented. Phase two can be achieved by using the well known **UNION-FIND** algorithm [11]. The main procedure is named **LABEL-CC**, and invoked with an array M and an integer N corresponding to the number of BLACK nodes in M . Steps 1 and 2 construct the **MAP** and initialize a list called **E-list** which will contain the equivalence pairs as they are generated. Procedure **PROPAGATE** implements phase one. It visits the nodes in M in ascending size order through **MAP**, explores the adjacencies between pairs of BLACK nodes by invoking **EXPLORE**, assigns labels produced by **ID-GENERATOR**, and accumulates equivalence pairs in the **E-list**. Procedure **EQ-NEIGHBOR** used by **EXPLORE** computes the key of $M[j]$'s equal-sized neighbor in the direction specified by the parameter *side*. The unspecified procedure **SEARCH**(M , P) works as follows: if P is a BLACK node then **SEARCH** returns an integer value k such that P is either equal to or contained in $M[k]$. However, if P is WHITE or GREY then **SEARCH** simply returns a zero. Unique labels are generated by procedure **ID-GENERATOR**, and assigned to BLACK nodes by procedure **ASSIGN-**

LABEL. Procedure UPDATE implements phase three by uniquely labeling each component while scanning M.

Procedure LABEL-CC(M, N)

```
begin
1 construct MAP;
2 E-list:={ $\emptyset$ };
3 PROPAGATE(M, N);
4 generate equivalence classes from E-list;
5 UPDATE(M, N);
end;
```

Procedure PROPAGATE(M, N)

```
begin
for i:=1 to N do
begin
j:=MAP[i];
for side in {N,E,S,W} do
if UNEXPLORED(M[j], side)
then EXPLORE(M[j], side);
if not LABEL(M[j]) then
M[j].ID:= ID-GENERATOR;
end;
end;
```

Procedure EXPLORE(M, j, side)

```
begin
neighbor:= EQ-NEIGHBOR(M[j], side);
k:= SEARCH(M, neighbor);
if k > 0 then
begin
mark OPSIDE(side) of M[k] "explored";
ASSIGN-LABEL(M[j], M[k]);
end;
end;
```

Procedure ASSIGN-LABEL(node,adj)

```
begin
if LABEL(node) and LABEL(adj)
then if node.ID  $\neq$  adj.ID
then add (node.ID,adj.ID) to E-list;
else if LABEL(node)
then adj.ID:=node.ID
else if LABEL(adj)
then node.ID:=adj.ID
else node.ID:=adj.ID:= ID-GENERATOR;
end;
```

Procedure UPDATE(M, N)

```
begin
for i:=1 to N do
M[i].ID:= FIND(M[i]);
end;
```

Example: As an example of the application of the algorithm, consider the region given in Fig. 2 whose block decomposition is given in Fig. 3. The BLACK nodes have been numbered in the order in which they were visited by phase one. Thus node 1 has been

visited before nodes 2, 3, etc. The labels assigned to the two components by the first phase of the algorithm are shown in Fig. 4. A short explanation about Fig. 4 is necessary at this point. When node 7 is visited, neither node 7 nor node 11, its eastern neighbor, has been labeled yet, thus label d is generated and assigned to both. When node 8 is visited, it has no label, but its northern neighbor, node 11, has already been assigned the label d, and thus node 8 is assigned the label d as well.

Fig. 4 illustrates the status of the image at the conclusion of the first phase of the algorithm. It has four different labels: a, b, c and d, with a equivalent to b, and b equivalent to c. The equivalence pair (a, b) was generated when node 9 was visited and its northern adjacency was explored. In essence, node 9 was labeled with a when node 1's western adjacency was explored, whereas node 10 was labeled with b when node 2's western adjacency was explored. Similarly, the equivalence pair (b,c) was generated when node 5 was visited.

Applying the second phase of the algorithm to the generated equivalence pairs results in the following two equivalence classes: {a,b,c} and {d}.

Fig. 5 shows the labels updated by the third phase of the algorithm.

Theorem 1: The time complexity of the connected component labeling algorithm is $O(n \cdot N)$.

Proof: Constructing the MAP requires time $O(N \cdot \log N)$. Phase one calls procedure EXPLORE N times, and procedure EXPLORE requires time $O(n + \log N)$, where n and log N originates from the invoking of procedure EQ-NEIGHBOR and SEARCH, respectively. Therefore phase one takes time $O(n \cdot N + N \cdot \log N)$. Phase two requires time $O(N \cdot \log N)$ [9]. Phase three requires time $O(N)$. Since $\log N < 2n$, the time complexity of the algorithm is therefore $O(n \cdot N)$.

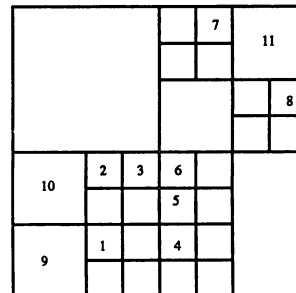


Fig. 3. Decomposition of Fig. 2.

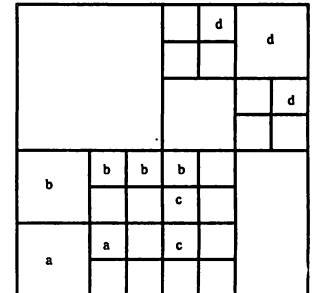


Fig. 4. Results of Phase 1.

6. COMPUTING THE PERIMETER

Perimeter computation is another basic operation in image processing. Algorithms computing the perimeter of a region in a binary image represented

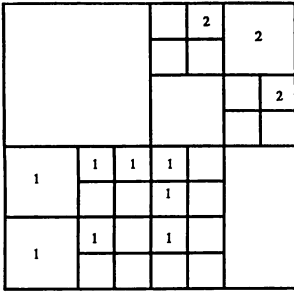


Fig. 5. Labels Resulting From Phase 5.

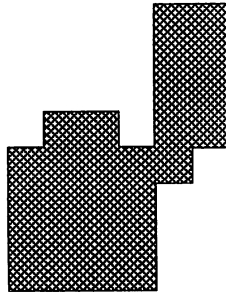


Fig. 6. A Connected Region.

either by an array or by a chain code are contained in [5]. An algorithm for computing the perimeter of a region encoded as a quadtree has also been developed by Samet [7].

The following perimeter computation algorithm traverses the MLQ in ascending size order. For each node *P* in the MLQ being visited, the length of each of its four sides is first included in the value of the perimeter. Then the neighbor nodes of *P* which have not been previously visited need to be considered. For each adjacent node *Q* that is BLACK, twice the length of the common side is deducted from the value of the perimeter. This reflects the fact that the segment between *P* and *Q* does not belong to the boundary of the region. The factor 2 occurs because the adjacency between two BLACK nodes is explored once and only once due to the traversal strategy used.

For example, given the BLACK node *D* in Fig. 7, the common segment between *D* and its southern neighbor *A* is explored by the time *D* is visited, but the same common segment is not considered when *A* is visited. Therefore the length of this segment *DA* has to be deducted in advance when *D* is visited.

The following procedure PERIMETER specifies the algorithm.

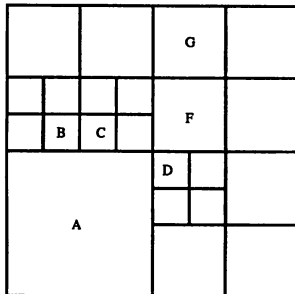


Fig. 7. Decomposition of Fig. 6.

Procedure PERIMETER(*M*, *N*)

begin

construct MAP;

perimeter:=0;

for *i*:=1 to *N*

begin

j:=MAP[*i*];

segment:=2** *M*[*j*].RES;

perimeter:= perimeter + 4 * segment;

for side in {N, E, S, W} do

if UNEXPLORED(*M*[*j*], side) then

begin

neighbor:= EQ-NEIGHBOR(*M*[*j*], side);

k:= SEARCH(*M*,neighbor);

if *k* > 0 then

begin

perimeter:=perimeter - 2 * segment;

mark OPSIDE(side) of *M*[*k*] "explored";

end;

end;

end;

return(perimeter);

end;

The key to this algorithm is that each node in the MLQ is visited once and, at most, its four neighbors need be explored. Such an advantage is achieved by traversing the MLQ in ascending size order. Otherwise, in the worst case, when the node being visited is of size 2^{n-1} by 2^{n-1} , 2^{n-1} nodes need be searched as in Samet's algorithm [7].

Example: Consider the region given in Fig. 6. The corresponding block decomposition is shown in Fig. 7. The MLQ contains six BLACK nodes representing blocks A, B, C, D, F and G. Assuming $n=3$, the perimeter is 30. Procedure PERIMETER visits the BLACK nodes in the order: B, C, D, F, G and A.

The following table contains a step-by-step trace through the algorithm for this example. The symbols 'φ' and '-' stand for don't care and non-existence, respectively.

Theorem 2: The time complexity of the algorithm PERIMETER is $O(n \cdot N)$.

Proof: Similar to the proof of Theorem 1.

Note that if the region is not connected, i.e., it contains more than one connected component, then the algorithm will return the sum of the perimeters of each connected component. It is, however, not difficult to compute the perimeter of every connected component of the region simultaneously in the same time complexity with a minor modification of the algorithm, provided that all connected components have been labeled.

node	side	neighbor	segment	contribut.	perim.
B	N	-	BC BA	4	4
	E	C		-2	4
	S	A		-2	2
	W	-			0
C	N	-	CA	4	4
	E	-			4
	S	A		-2	4
	W	ϕ			2
D	N	F	DF	4	6
	E	-		-2	4
	S	-			4
	W	A	DA	-2	4
F	N	G	FG	8	10
	E	-		-4	6
	S	ϕ			6
	W	-			6
G	N	-		8	14
	E	-			14
	S	ϕ			14
	W	-			14
A	N	ϕ		16	30
	E	ϕ			30
	S	-			30
	W	-			30

7. CONCLUSION

Techniques for labeling connected components and computing the perimeter of a region have been described. The algorithm for labeling connected components is superior to the one using a standard linear quadtree [3] in the sense that it is capable of handling regions with arbitrary configurations. By the same token, the perimeter computation algorithm shares the same advantage.

REFERENCES

1. Davis, W.A., and Wang, X., "A New Approach to Linear Quadrees", *Proceedings Graphics Interface '85*, pp. 195-202, Montreal, May 1985.
2. Gargantini, I., "An efficient way to represent properties of quadrees", *Comm. ACM*, Vol. 25, pp. 905-910, Dec. 1982.
3. Gargantini, I., "Detection of Connectivity for Regions Using Linear Quadrees", *Comp. & Math. with Appl.*, Vol. 8, pp. 319-327, 1982.
4. Rosenfeld, A., "Connectivity in Digital Pictures", *J.ACM*, Vol. 17, pp. 146-160, 1970.
5. Rosenfeld, A. and Kak, A.C., "*Digital Picture Processing*", Academic Press, New York, 1976.
6. Samet, H., "Connected Component Labeling Using Quadrees", *J.ACM*, Vol. 28, pp. 487-501, 1981.
7. Samet, H., "Computing Perimeters of Regions in Images Represented by Quadrees", *IEEE Trans. Pattern Analy. & Mach. Intel.*, Vol. PAMI-3, pp. 683-687, 1981.
8. Samet, H., "Neighbor Finding Techniques for Images Represented by Quadrees", *Comput. Graphics and Image Process.*, Vol. 18, pp. 37-57, 1982.
9. Wang, X., "Some New Approaches for Linear Quadrees", M.Sc. Thesis, Department of Computing Science, University of Alberta, 1985.
10. Klinger, A. and Dyer, C.R., "Experiments in Picture Representation Using Regular Decomposition", *Computer Graphics and Image Processing*, Vol. 5, pp. 68-105, 1976.
11. Aho, A., Hopcroft, J. and Ullman, J.D., "*The Design and Analysis of Computer Algorithms*", Addison-Wesley, Reading, Mass., 1974.

ASTERISK*: AN EXTENSIBLE TESTBED FOR SPLINE DEVELOPMENT

Jonathan R. Gross †

Tony D. DeRose ‡

Brian A. Barsky

Berkeley Computer Graphics Laboratory

Computer Science Division

Department of Electrical Engineering and Computer Sciences

University of California

Berkeley, CA. 94720

U.S.A.

Abstract

*Asterisk** is a testbed system designed to support the development of new kinds of splines. The key concept is the integration of *symbolic computation facilities* with tools for *interactively modifying and comparing* different splines. By modeling a spline as a list of attributes, *Asterisk** can be used to create and manipulate almost any spline, without making assumptions about the future directions of spline research.

Résumé

*Asterisk** est un système d'étude créé pour appuyer le développement de nouvelles sortes de courbes à base de splines. Le concept clef est l'intégration d'*outils interactifs pour modifier et comparer* différents courbes, avec un *système de calcul symbolique*. En décrivant une courbe par une liste d'attributs, le système *Asterisk** peut être utilisé pour créer et manipuler presque n'importe courbe à base de splines sans présumer les directions de cette recherche à l'avenir.

KEYWORDS: geometric modeling, software, symbolic computation, parametric curves.

1. Introduction and Motivation

A *spline* is a mathematical formulation of a curve used for a wide variety of modeling applications. Unlike polygonal representations, splines provide compact and resolution-independent descriptions of complex objects. Many splines combine a set of *control vertices* with a set of *blending functions* to determine the path of a curve through space. Different splines have different properties, and the choice of which spline to use depends on the problem at hand. For example, some splines *interpolate* (pass through) the control vertices, while others *approximate* (pass near) them.

There is no single type of spline that is ideal for all applications; each spline is a tool best suited to some particular set of tasks. Current spline research involves developing new splines with specific properties as solutions to different problems. To create a spline representation with a desired set of properties, the blending functions must satisfy a set of *constraints*. Sometimes this involves combining constraints from existing splines to arrive at a new formulation [4]; in other cases it requires substantial trial and error to find a suitable set of constraints that produces the desired properties.

Most splines can be described mathematically in simple terms. In spite of this, establishing the constraints that correspond to desired properties, and then solving for a set of blending functions that satisfy those constraints is a task that can easily become overwhelming. Computer algebra systems such as *Vaxima* [5] can perform error-

* A Simple Testbed for the Evaluation and Rendering (Interactively) of Splines of many Kinds.

† Author's current address: Vertigo Systems International, Suite 221-119 West Pender Street, Vancouver, B.C. Canada V6B 1S5.

‡ Author's current address: Department of Computer Science, FR-35, University of Washington, Seattle, WA. 98195, U.S.A.

This work was supported in part by the Defense Advanced Research Projects Agency under contract number N00039-82-C-0235, the National Science Foundation under grant number ECS-8204381, the Natural Sciences and Engineering Research Council of Canada, the State of California under a Microelectronics Innovation and Computer Research Opportunities grant, and a Shell Doctoral Fellowship.

free symbolic (as opposed to numeric) computations of complexity too great for humans to handle. Vaxima takes a description of the constraints and produces a *symbolic representation* of the blending functions. Recent efforts in spline development have taken advantage of such capabilities [1,4].

To understand the behaviour of new splines, it is useful to draw them, preferably in an interactive setting. This generally requires an *evaluation routine* to compute points on the curve, and a collection of routines to graphically display and modify the curve. The coding of the evaluation routine requires tedious translation from the symbolic representation into the implementation language.

Intuition gained from visual feedback, as well as comparison with other spline representations, often leads to modification of the constraints that define the spline. This in turn changes the symbolic representation, necessitating a recoding of the evaluation routine.

We would like to tighten this specification/visualization loop and automate the translation step. Feng & Riesenfeld [6] describe just such a system for the development of *Boolean sum surfaces*. Although Feng & Riesenfeld had access to the symbolic algebra system *REDUCE* [7], they chose not to use it for two reasons: they were interested primarily in simple operations on rational bivariate polynomials, and they felt that they could not provide a sufficiently high level of user interaction by running a large system like REDUCE. Instead, Feng & Riesenfeld implemented a small, specialized algebra system capable of performing operations such as operator composition, evaluation, and symbolic differentiation of bivariate polynomials.

This paper describes a testbed system called *Asterisk** designed to integrate the power of symbolic computation with convenient, extensible, interactive tools for modifying and comparing different splines [8]. *Asterisk** is intended for use by those doing research in spline techniques rather than those wishing to design specific objects using splines. Our system is similar to that of Feng & Riesenfeld, but differs in that the emphasis is on *generality*. Since future spline representations may not conform to all the paradigms of current spline techniques, one of our primary goals was to avoid making too many assumptions about the future directions of spline research.

*Asterisk** supports generality in three ways: the use of a complete symbolic algebra system (Vaxima), an extensible model of a spline (a list of attributes), and an extensible set of interactive spline management and graphics routines (written in Lisp). The blending functions may be developed and verified within Vaxima, and then used directly to draw or plot the resulting curves. As new algorithms are developed to manipulate the spline representations, routines to implement these algorithms can be written quickly, thus extending the basic building blocks that *Asterisk** provides. In this way, spline researchers

can adapt the interactive testbed to their particular requirements and preferences.

2. The Asterisk* Solution

2.1. The Asterisk* Model of a Spline

Throughout this paper we use the term spline to mean a piecewise parametric function of the form

$$Q(u) = \sum_{i=0}^m V_i B_i(u) \quad (2.1)$$

where the V_i represent a set of $m + 1$ control vertices and the $B_i(u)$ represent a set of blending functions.

*Asterisk** provides a uniform definition protocol for spline specification by modeling a spline as a list of attributes. Each attribute is a <name,value> pair. Table 1 summarizes some common attributes. In Lisp, these are stored as *property lists* of the spline's name. For instance, the attribute list for a spline named "ucb_spline" is stored on the property list for the symbol *ucb_spline*. The attribute list in Figure 1 was used to define the spline illustrated in Colour Plate 1.

```
(splinename ucb_spline
  polycolor GREEN
  polystyle DASHED
  polydisplay ON
  curvecolor WHITE
  curvestyle SOLID
  curvedisplay ON
  controlpolygon ((.42 .41) (.41 .73) (.55 .86)
                 (.86 .76) (.86 .59) (.61 .58)
                 (.61 .38) (.83 .35))
  evalroutine ucb_eval
  parameterstep 0.0625)
```

Figure 1: The attribute list for *ucb_spline*.

2.2. System Architecture

Conceptually, the *Asterisk** testbed consists of three logical parts:

- A *Vaxima to Lisp translator* for automatically generating evaluation routines from Vaxima descriptions. This is the transition from the analytic expressions for the blending functions (the symbolic representation), to the procedures that evaluate the blending functions (the numeric representation required for graphical display).
- *Interface routines* for support of graphical interactions such as geometric input or display of a curve. This module provides a device independent graphical interface to the rest of the system.

Attribute Name	Value Type	Purpose
controlpolygon	List of vertices V_i	Rough specification of curve path.
evalroutine	Name of a function	Names the function which evaluates points on the curve.
parameterstep	Real number	The amount the domain parameter u is varied between consecutive calls to the evaluation routine.
polystyle (curvestyle)	DOTTED DASHED SOLID	Style used to draw control polygon (curve)
polycolor (curvecolor)	Colour name	Colour used to draw control polygon (curve)
polydisplay (curvedisplay)	ON OFF	Should polygon (curve) be displayed?
degree	Integer	The polynomial degree of the curve.
knotvector	List of real numbers u_i	Specifies a set of parameter values associated with the break-points between curve segments. This attribute is useful for B-spline curves and cubic interpolatory splines (cf. Bartels <i>et al</i> [3]).
beta1 (beta2)	Real number	Shape parameters for geometrically continuous techniques such as Beta-splines [1,2].

Table 1: Common Attributes

- *Spline management routines* for manipulating attribute lists. The spline management routines facilitate convenient modification of the spline attributes. These routines invoke the interface routines to interact with the user and invoke the evaluation routines to generate curve segments for each spline.

When a *draw curve* operation is initiated, the spline management routines repeatedly invoke the evaluation routine for the spline, passing in a value of the parameter. The parameter step attribute determines which, and how many, parameter values are passed in. It is the responsibility of the evaluation routine to return the point on the curve corresponding to each given parameter value. This is done by referring to the specific attributes that are needed — attributes which do not affect the computation are ignored. The spline management routines invoke the interface routines to connect the points on the curve into a piecewise linear approximation of the curve.

Ideally, we would like the spline management and interface routines running within the Vaxima environment. This approach was taken in an early implementation; unfortunately, the code space requirements of Vaxima caused excessive page faulting, resulting in poor performance. This problem can be avoided by executing Vaxima and the Vaxima to Lisp translator in one process, and the spline management and interface routines in another. In the current implementation these processes run on separate machines and communicate via disk files transmitted over a local area network. An advantage of the two-process structure is the ability to substitute an alternate symbolic computation system, such as REDUCE, by supplying an appropriate translation routine.

2.3. Extension Features

*Asterisk** supports extensibility in two ways. Both the set of functions that manipulate splines and the data structures used to represent splines can be extended as needed.

Functional extension is a byproduct of the Lisp environment. Extension of spline representations follows from the fact that each evaluation routine ignores attributes which do not affect its computation. Thus, not all splines require all attributes, and new attributes can easily be added for evaluation routines which require them. More importantly, these additions are completely transparent to other evaluation routines, meaning that existing code need not be modified.

3. Applications

3.1. Comparisons of Spline Curves

It is often desirable to compare and contrast the behaviour of various curves. For instance, one might want to determine the effect of changing the control polygon, the shape parameters, or the polynomial degree of a curve. One may also wish to compare the shapes of two curves of different types defined by the same control polygon. In *Asterisk**, all of these comparisons can be accomplished using the following three steps:

1. Define the curve that is to be the basis of the comparison.
2. Make a copy of the curve by copying its attribute list.

3. Change one (or more) of the new curve's attributes.

To visually distinguish between the two curves it is often useful to change display attributes such as the line style or colour of the control polygon or curve. Colour Plates 1 through 4 illustrate such comparisons.

3.2. The Development of a New Curve Type

As mentioned in Section 1, one of the main motivations for *Asterisk** was the ability to easily define and assess new curve types in an interactive setting. The process of designing new curve types typically involves repeated iteration through the following steps:

1. Determine the desired properties for the curve.
2. State these properties as mathematical constraints.
3. Use *vaxima* to solve the constraints.
4. Interactively assess and experiment with the curve.

As an example, Table 2 describes a new curve that was constructed and tested using *Asterisk**. The three columns of the table contain the desired properties for the curve (step 1), the corresponding mathematical constraints (step 2), and the associated *Vaxima* expressions (step 3). In the bottom right section of the table, the lengthy derivation of the *Vaxima* expressions describing the convex hull property

has been omitted for the sake of brevity. Figure 2 contains the Lisp evaluation routine produced by the *Vaxima* to Lisp translator.

To interactively assess the new technique (step 4), we create a new spline attribute list and graphically specify a control polygon to which the evaluation routine is applied (see Colour Plate 3). The techniques of Section 3.1 can then be used to observe the behaviour of the curve under various conditions.

3.3. Interactive Testing of Algorithms

To this point, we have concentrated on the specification and comparison of splines themselves. As mentioned above, this involves creation and modification of spline attribute lists, using various operations which *Asterisk** provides. Another aspect of current spline research is the development of algorithms that operate on splines. To test interactively such an algorithm without the benefit of a testbed system such as *Asterisk**, one would be forced to write a specialized program to manage data structures and user-interaction, in addition to performing the required computation.

With *Asterisk**, one is able to concentrate on writing a Lisp function to implement the new algorithm alone; data structure manipulation is handled by the spline

Property	Constraints	Vaxima Expression
Cubic polynomial curve defined by 4 control vertices	$Q(u) = \sum_{i=0}^3 V_i B_i(u)$ <p>where</p> $B_i(u) = \sum_{j=0}^3 k_{ij} u^j.$	<pre>poly(a,b,c,d) := a+b*u+c*u^2+d*u^3; b0(u) := '(poly(k00,k01,k02,k03)); b1(u) := '(poly(k10,k11,k12,k13)); b2(u) := '(poly(k20,k21,k22,k23)); b3(u) := '(poly(k30,k31,k32,k33));</pre>
<i>Symmetry</i> : reversing the control points reverses the curve.	$B_0(u) = B_3(1-u)$ $B_1(u) = B_2(1-u)$	<pre>b2(u) := '(b1(1-u)); b3(u) := '(b0(1-u)); unknowns : [k00,k01,k02,k03, k10,k11,k12,k13];</pre>
Interpolation of first and last control vertices and the midpoint of the center leg of the control polygon.	$Q(0) = V_0$ $Q\left(\frac{1}{2}\right) = \frac{V_1 + V_2}{2}$ $Q(1) = V_3$	<pre>e0:b0(0)=1; e1:b0(1/2)=0; e2:b0(1)=0; e3:b1(0)=0; e4:b1(1/2)=1/2; e5:b1(1)=0;</pre>
<i>Convex hull</i> : the curve should lie entirely within the smallest convex region containing the control polygon.	$\sum_{i=0}^3 B_i(u) = 1$ $B_i(u) \geq 0, u \in [0, 1]$	<pre>b0d1(u) := '(diff(b0(u),u)); b1d1(u) := '(diff(b1(u),u)); e6:b0d1(0) = -1; e7: b1d1(0)=0; equations : [e0,e1,e2,e3,e4,e5,e6,e7]; answer : linsolve(equations, unknowns);</pre>

Table 2: Definition of a New Curve Type

```

(def b0 (lambda (u)
  (+ 1.0 (* -5.0 u)
    (* 8.0 (expt u 2.0))
    (* -4.0 (expt u 3.0)))))

(def b1 (lambda (u)
  (+ (* 4.0 u)
    (* -8.0 (expt u 2.0))
    (* 4.0 (expt u 3.0)))))

(def b2 (lambda (u)
  (+ (* 4.0 (expt u 2.0))
    (* -4.0 (expt u 3.0)))))

(def b3 (lambda (u)
  (+ u (* -4.0 (expt u 2.0))
    (* 4.0 (expt u 3.0)))))

```

Figure 2: Code produced by the *vaxima* to lisp translator for the example of Table 2.

management routines and user interaction is handled by the interface routines. If the operation provided by the algorithm is of lasting interest, it is a simple matter to include it on one of the *Asterisk** menus, thereby allowing the user to graphically invoke the algorithm by selecting the menu item and the spline that is to be operated on.

As a specific example, consider an algorithm to perform linear subdivision of polynomial splines. The subdivision process takes as input the control polygon describing one segment of a curve, and returns a control polygon that generates only a portion of the original curve (see Colour Plate 4). This operation was added to *Asterisk** by writing a Lisp function that requests a copy of a spline's attribute list, and replaces the *controlpolygon* attribute with a subdivision polygon. The new operation can now be applied interactively and the subdivided curves can be directly compared to the original ones.

4. Summary

Symbolic algebra systems have allowed increasingly complex spline formulations to be developed. However, interactive visual feedback is needed to understand the behaviour of the resulting curves. Intuition thus gained often leads to reformulation of the underlying mathematics.

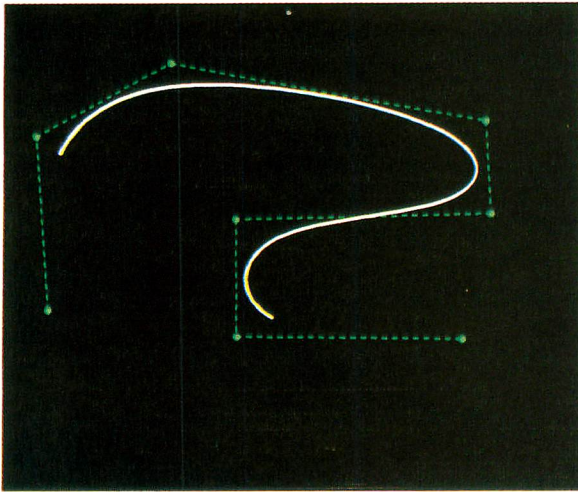
*Asterisk** has been developed to tighten this design loop by automating the conversion between symbolic and graphical representations. As such, it provides a framework for the interactive development, manipulation, and comparison of new splines; it is not merely a program with a fixed set of built-in splines. *Asterisk** also provides facilities for quickly implementing and exercising new algorithms that operate on spline representations. In addition, because *Asterisk** allows one to concentrate on the mathematics and implementation of computational algorithms without worrying about user interaction and

graphical display, we believe that it will prove useful as a tool for teaching basic curve and surface techniques.

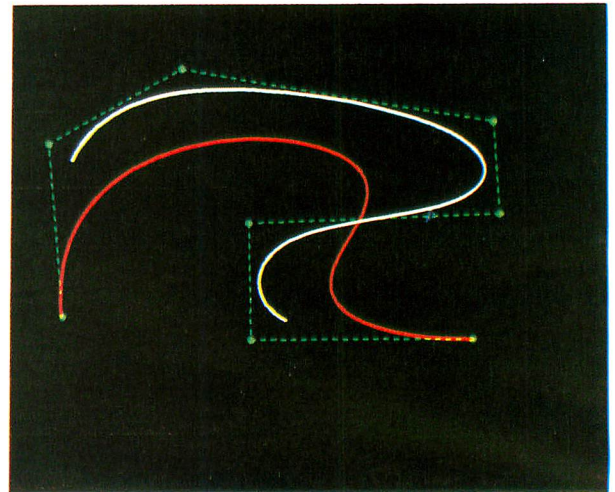
The *Asterisk** model of a spline as a list of attributes does not restrict us to dealing with two-dimensional curves. Modifying the data structures and graphical display routines to handle three-dimensional curves and surfaces is an obvious extension.

References

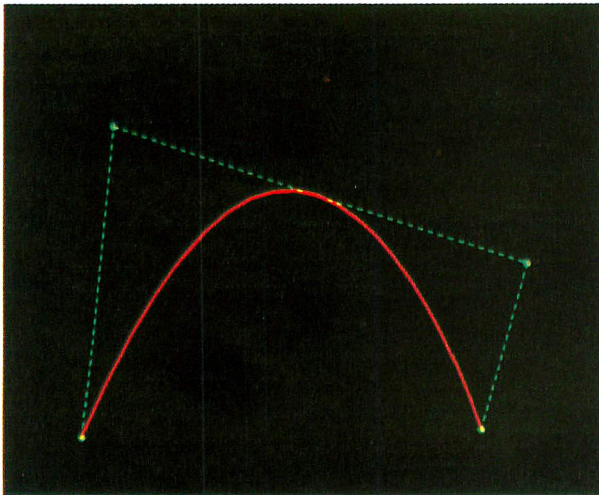
1. Brian A. Barsky, *The Beta-spline: A Local Representation Based on Shape Parameters and Fundamental Geometric Measures*, Ph.D. Thesis, University of Utah, Salt Lake City, Utah (December, 1981).
2. Brian A. Barsky, *Computer Graphics and Geometric Modelling Using Beta-splines*, Springer-Verlag, Tokyo (1986).
3. Richard H. Bartels, John C. Beatty, and Brian A. Barsky, *An Introduction to the Use of Splines in Computer Graphics*, UCB/CSD 83/136, Computer Science Division, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, California, USA (August, 1983). Also Tech. Report No. CS-83-9, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.
4. Tony D. DeRose and Brian A. Barsky, "Geometric Continuity and Shape Parameters for Catmull-Rom Splines (Extended Abstract)," pp. 57-62 in *Proceedings of Graphics Interface '84*, Ottawa (27 May - 1 June, 1984).
5. Richard J. Fateman, *Addendum to the MACSYMA Reference Manual for the VAX*, Computer Science Division, University of California, Berkeley (1982).
6. David Y. Feng and Richard F. Riesenfeld, "A Symbolic System for Computer-Aided Development of Surface Interpolants," *Software — Practice and Experience*, Vol. 8, No. 4, July-August, 1978, pp. 461-481.
7. Martin L. Griss, *A REDUCE Symbolic-Numeric Tutorial*, Utah Symbolic Computation Group Operating Note, Technical Report No. UCP-32, Department of Computer Science, University of Utah (October, 1977).
8. Jonathan R. Gross, *Software Tools for Computer Graphics Research and Development*, Masters Report, University of California, Berkeley (May, 1984).



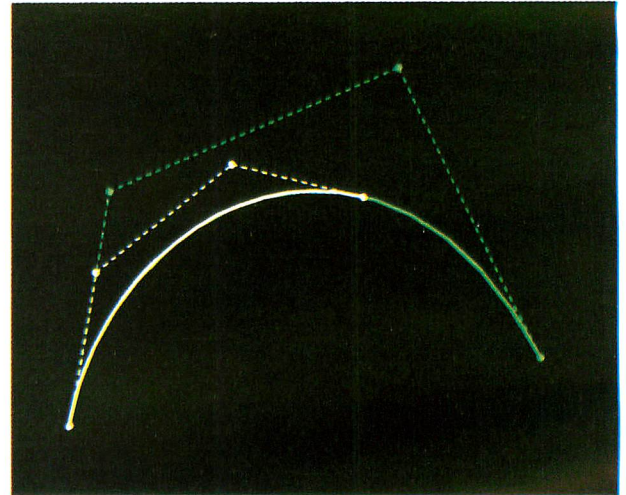
Colour Plate 1: A uniform cubic B-spline. The control polygon is coloured green and the curve is coloured white.



Colour Plate 2: Two splines that share a common control polygon (shown in green), but have different evaluation routine attributes. The white curve is a uniform cubic B-spline, and the red curve is a seventh degree Bézier curve.



Colour Plate 3: This plate illustrates a midpoint curve (shown in red), together with its control polygon (shown in green). As required by the representation, the curve interpolates the first and last control vertices, as well as the midpoint of the middle leg of the control polygon.



Colour Plate 4: This plate demonstrates the behaviour of a subdivision algorithm for a cubic Bézier curve. The original control polygon (shown in green) generates the green curve when its parameter varies on the interval $[0, 1]$. The subdivision polygon (shown in white) generates the white curve when its parameter varies on the interval $[0, 1]$. The subdivision polygon is constructed so that the white curve is identical to the portion of green curve corresponding to the interval $[0, 2/3]$.

GRAPHICAL APPLICATIONS OF L-SYSTEMS

Przemysław Prusinkiewicz

Department of Computer Science
University of Regina
Regina, Saskatchewan, CANADA S4S 0A2

ABSTRACT.

A new method for generating pictures is presented and illustrated with examples. The idea is to generate a string of symbols using an L-system, and to interpret this string as a sequence of commands which control a "turtle". Suitable generalizations of the notions of the L-system and of a turtle are introduced. The resulting mathematical model can be used to create a variety of (finite approximations of) fractal curves, ranging from Koch curves, to classic space-filling curves, to relatively realistic-looking pictures of plants and trees. All these pictures are defined in a uniform and compact way.

RESUME.

Une nouvelle méthode pour engendrer des images est présentée et illustrée avec des exemples. Cette méthode comprend deux étapes. On commence par engendrer une séquence de symboles avec un L-système. Ensuite on utilise cette séquence pour contrôler les mouvements d'une tortue qui trace l'image en question. Les notions d'un L-système et d'une tortue sont généralisées pour mieux correspondre au but de la création des images. Le modèle mathématique résultant s'applique à la création d'une large variété d'objets fractals, y compris des courbes de Koch, des courbes qui remplissent tout une aire plane, ainsi que des images relativement réalistes des plantes et des arbres. Toutes ces images sont définies d'une manière homogène et compacte.

KEYWORDS: L-systems, turtle geometry, fractals, space-filling curves, plants, trees.

1. INTRODUCTION.

Rewriting systems can be used to generate pictures in two different ways. In the first case, rewriting systems operate directly on two-dimensional objects, such as arrays [Kirsch 1964, Dacey 1970], graphs [Rosenfeld and Milgram 1972, Pfaltz 1972], or "shapes" [Gips 1975, Stiny 1975]. In the second case, string grammars (in the broad sense of the word, including parallel rewriting systems) are used to define strings of symbols. A graphic interpretation function subsequently maps these strings into pictures. This paper describes a method for generating pictures based on this second approach. After the idea of applying string grammars to pictures is put into a historic perspective in Section

2, attention is focused on L-systems [Lindenmayer 1968]. The necessary definitions related to OL-systems are collected in Section 3. Section 4 adapts the notion of a "turtle" [Papert 1980, Abelson and diSessa 1982] to the purpose of graphical interpretation of strings, and presents examples of pictures generated by OL-systems under this interpretation. Section 5 extends this basic approach in two directions: by generalizing the notion of the L-system, and by increasing the range of string symbols interpreted by the turtle. Section 6 presents conclusions and lists several open problems.

2. THE HISTORICAL BACKGROUND.

The idea of describing pictures using formal (string) languages emerged a few years after Chomsky established the fundamental concept of a phrase-structure grammar. Narasimhan [1962, 1966] and Ledley [1964, 1965] are credited with the first results in this area. Their interest was in the recognition of handwritten characters and chromosomes, respectively. An approach designed for describing a wider class of pictures using string grammars was proposed by Shaw [1969]. For a survey of these early results, see Fu [1980].

The early research concentrated on picture recognition. Pictures were described as strings of symbols which represented selected primitives, such as straight segments, sharp V-turns, wide U-turns or branches. In some cases, relations between picture elements, such as ABOVE, BELOW, of INSIDE, were also considered as primitives. The actual picture recognition was performed by parsing the resulting strings.

In the case of picture generation, the correspondence between string symbols and picture primitives must be specified in more detail. The first such specification, known as chain coding, was developed by Freeman [1961]. Feder [1968] showed that the languages of chain codes describing such classes of figures as straight lines of arbitrary slope, circles of arbitrary radius, or convex figures in a plane, are all context sensitive. It was subsequently pointed out (for example, by Fu [1980]) that even intuitively simpler classes of pictures, for example the set of all rectilinear squares in an integer grid, correspond to context-sensitive chain-code languages. To a certain degree, this discouraged a further study of chain-code languages, for the context-sensitive grammars are usually difficult to construct and do not provide an intuitively clear description of languages. Neverthe-

less, picture generation using Chomsky grammars and the chain interpretation has recently received considerable attention [Maurer, Rozenberg and Welzl 1982, Sudborough and Welzl 1985].

In order to describe growth of living organisms, Lindenmayer [1968] introduced the notion of a parallel rewriting system. The Lindenmayer systems, or L-systems, attracted the interest of many researchers, and the theory of L-systems was soon extensively developed [Herman and Rozenberg 1975, Lindenmayer and Rozenberg 1976]. However, although a geometrical interpretation of strings was at the origin of L-systems, they were not applied to picture generation until 1984, when Aono and Kunii [1984], and Smith [1984] used them to create realistic-looking images of trees and plants. Approximately at the same time, Siromoney and Subramanian [1983] noticed that L-systems with chain-code interpretation could be used to generate some space-filling curves.

This paper further investigates graphical applications of L-systems. The emphasis is on the turtle interpretation rather than the chain coding, because the turtle interpretation allows for generating pictures which are not confined to a grid. The notions of the L-system and of a turtle are generalized to provide increased flexibility in picture specification. The resulting mathematical model is used to generate a wide spectrum of fractal curves.

3. 0L-SYSTEMS.

This section summarizes fundamental definitions and notations related to 0L-systems. For their tutorial introduction, see Salomaa [1973], and Herman and Rozenberg [1975].

Let V denote an alphabet, V^* - the set of all words over V , and V^+ - the set of all nonempty words over V .

Definition 3.1. A **0L-system** is an ordered triplet $G = \langle V, \omega, P \rangle$ where V is the **alphabet** of the system, $\omega \in V^+$ is a nonempty word called the **axiom** and $P \subset V \times V^*$ is a finite set of **productions**. If a pair (a, χ) is a production, we write $a \rightarrow \chi$. The letter a and the word χ are called the **predecessor** and the **successor** of this production, respectively. It is assumed that for any letter $a \in V$, there is at least one word $\chi \in V^*$ such that $a \rightarrow \chi$. A 0L-system is **deterministic** iff for each $a \in V$ there is exactly one $\chi \in V^*$ such that $a \rightarrow \chi$.

Definition 3.2. Let $G = \langle V, \omega, P \rangle$ be a 0L-system, and suppose that $\mu = a_1 \dots a_m$ is an arbitrary word over V . We will say that the word $v = \chi_1 \dots \chi_m \in V^*$ is **directly derived** from (or **generated** by) μ and write $\mu \Rightarrow v$ iff $a_i \rightarrow \chi_i$ for all $i = 1, \dots, m$. A word v is generated by G in a derivation of **length** n if there exists a sequence of words $\mu_0, \mu_1, \dots, \mu_n$ such that $\mu_0 = \omega$, $\mu_n = v$ and $\mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$.

4. GENERATING PICTURES USING 0L-SYSTEMS WITH TURTLE INTERPRETATION.

This sections formalizes the notion of the turtle interpretation of a string and provides examples of pictures generated by 0L-systems under this interpretation.

Definition 4.1. A **picture** Π is a set of points in the plane: $\Pi \in 2^{R \times R}$. A function $I: V^* \rightarrow 2^{R \times R}$ mapping the set of strings

over the alphabet V into the set of pictures is called a (graphic) **interpretation function**.

Definition 4.2. A **state** of the turtle is a triplet (x, y, α) , where the coordinates (x, y) represent the turtle's **position**, and angle α , called the turtle's **heading**, is interpreted as the direction in which the turtle is facing. Given the **step size** d and the **angle increment** δ , the turtle can respond to commands represented by the following symbols:

- F** Move forward a step of length d . The state of the turtle changes to (x', y', α) , where $x' = x + d \cos \alpha$ and $y' = y + d \sin \alpha$. A line segment between points (x, y) and (x', y') is drawn.
- f** Move forward a step of length d without drawing a line.
- +** Turn right by angle δ . The next state of the turtle is $(x, y, \alpha + \delta)$. (Here we assume that the positive orientation of angles is clockwise.)
- Turn left by angle δ . The next state of the turtle is $(x, y, \alpha - \delta)$.
- |** Turn away. The state of the turtle changes to $(x, y, \alpha + 180^\circ)$.

All other symbols are ignored by the turtle (the turtle preserves its state).

Definition 4.3. Let v be a string, (x_0, y_0, α_0) - the initial state of the turtle, and d, δ - fixed parameters. The picture (set of lines) drawn by the turtle responding to the string v is called the **turtle interpretation** of v .

Figure 1 presents chain interpretations of the words generated by some deterministic 0L-systems. In all cases, the angle increment δ is equal to 90° . Data under each picture indicate the length of derivation n , the step size d (in pixels), the axiom ω of the 0L-system, and the set of productions P . Note the variety of the shapes obtained, and the simplicity of the underlying L-systems.

The Hilbert curve (Fig. 1f) is representative of classic space-filling curves. Other well-known space-filling curves were discovered by Peano [1890] and Sierpiński [1912]. The Peano curve is generated by the L-system with axiom **X** and productions:

$$\begin{aligned} X &\rightarrow XFYFX + F + YFXFY - F - XFYFX \\ Y &\rightarrow YFXFY - F - XFYFX + F + YFXFY \\ F &\rightarrow F \quad + \rightarrow + \quad - \rightarrow - \end{aligned}$$

A square-grid approximation of the Sierpiński curve is generated by the L-system with axiom **F+XF+F+XF** and productions:

$$\begin{aligned} X &\rightarrow XF - F + F - XF + F + XF - F + F - X \\ F &\rightarrow F \quad + \rightarrow + \quad - \rightarrow - \end{aligned}$$

Once again, note the simplicity of these descriptions. For the long history of creating possibly elegant and compact programs for generating space-filling curves, see [Null 1971, Wirth 1976, Goldschlager 1981, Witten and Wyvill 1983, Cole 1983, Cole 1985].

5. EXTENSIONS OF THE BASIC MODEL.

This section generalizes the notion of the L-system and extends the notion of the turtle interpretation of a string. The purpose is to increase the flexibility of picture specification.

Various extensions to OL-systems have been thoroughly studied in the past [Salomaa 1973, Herman and Rozenberg 1975, Lindenmayer and Rozenberg 1976]. Specifically, **2L-systems** use productions of the form $a_l < a > a_r \rightarrow \chi$; this notation means that the letter a (called the **strict predecessor**) can produce word χ iff a is preceded by letter a_l and followed by a_r . Thus, letters a_l and a_r form the left and the right **context** of a in this production. Productions in **1L-systems** have one-side context only; consequently, they are either of the form $a_l < a \rightarrow \chi$ or $a > a_r \rightarrow \chi$. OL-systems, 1L-systems and 2L-systems belong to a wider class of **(k,l)-systems**. In a **(k,l)-system**, the left context is a word of length k , and the right context is a word of length l . How-

ever, the strict predecessor is still limited to a single letter.

For the purpose of picture generation it is convenient to generalize L-systems even further by allowing for productions of the form $\eta_l < \eta > \eta_r \rightarrow \chi$, where all three components of the predecessor are words of arbitrary length. This modification of L-systems seriously affects the notion of the direct derivation of words. In OL-systems and **(k,l)-systems** the strict predecessors of all productions consist of one letter. Consequently, in a derivation $\mu \Rightarrow v$ each letter of μ is a strict predecessor of some production. On the other hand, if the lengths of the strict predecessors η may vary, the partition of μ into strict predecessors depends on the particular productions used to derive v . The method for parti-

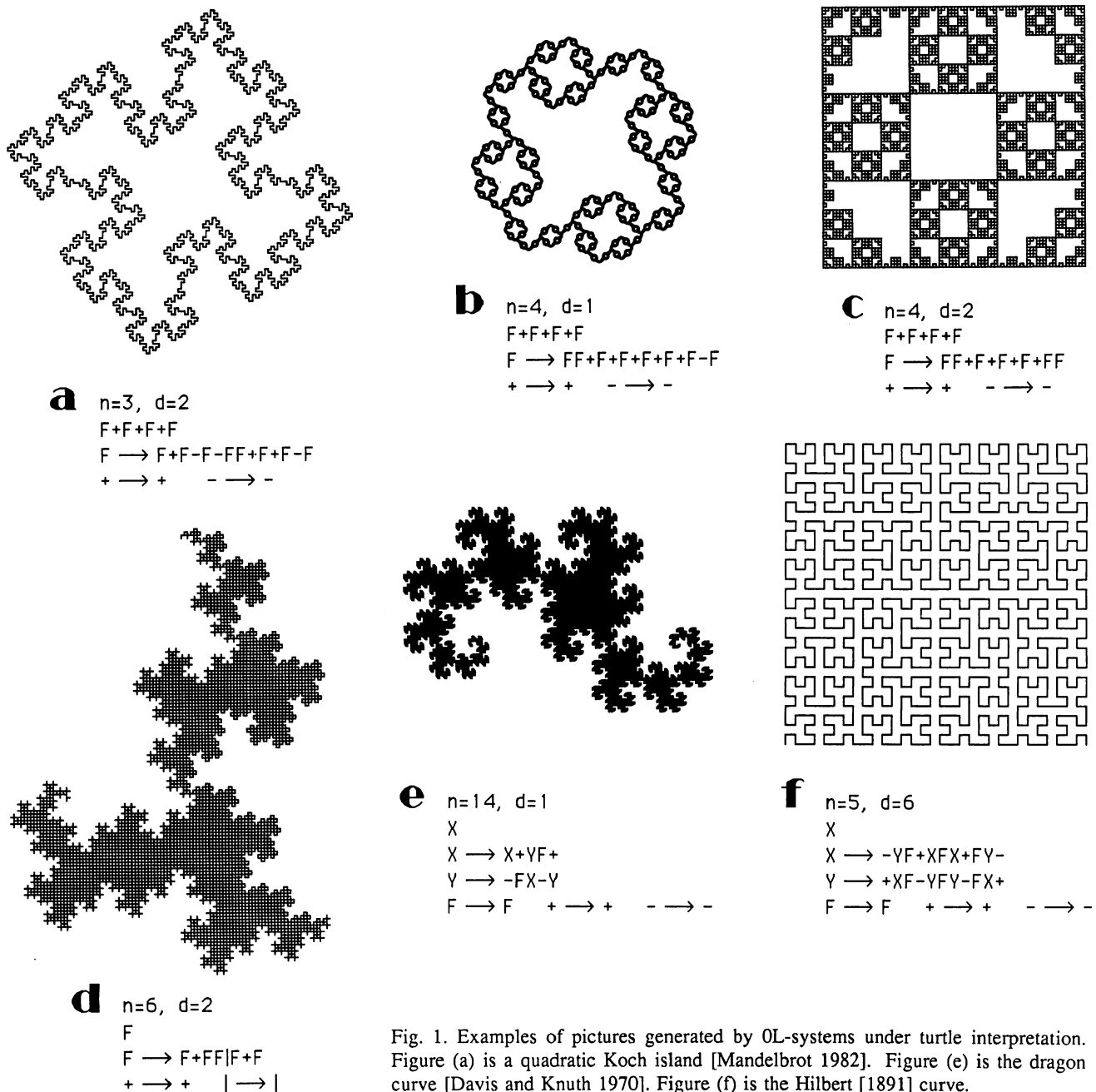


Fig. 1. Examples of pictures generated by OL-systems under turtle interpretation. Figure (a) is a quadratic Koch island [Mandelbrot 1982]. Figure (e) is the dragon curve [Davis and Knuth 1970]. Figure (f) is the Hilbert [1891] curve.

tioning the word μ introduced below is based on scanning the string μ from left to right.

Definition 5.1. A **pL-system** (pseudo-L-system) is an ordered triplet $G = \langle V, \omega, p \rangle$, where V is the alphabet of the system, $\omega \in V^+$ is the axiom, and $p: \{1, \dots, N\} \rightarrow (V^* \times V^+ \times V^*) \times V^*$ is an **ordered set of productions**. Instead of $p(i) = (\eta_l, \eta, \eta_r, \chi)$ we write that production $p(i)$ is equal to $\eta_l < \eta > \eta_r \rightarrow \chi$. Let l' , c' and r' denote the lengths of the strings η_l , η and η_r , respectively. Production p_i **matches** string $\mu = a_0 \dots a_m$ at position s , $0 \leq s \leq m$, iff the string μ can be represented as $a_0 \dots a_{s-l'-1} \eta_l \eta \eta_r a_{s+c'+r'} \dots a_m$. The production p_i is **applicable** to the string μ at position s iff it matches μ at position s and no production p_j preceding p_i ($j < i$) matches μ at this same position.

Note. In order to keep specifications of L-systems short, it is convenient to assume that all productions of the form $a \rightarrow a$, $a \in V$ are automatically appended at the end of any set p . Consequently, it is not necessary to list these productions when specifying p . On the other hand, any production of the form $a \rightarrow \chi$ entered explicitly into the set p will precede $a \rightarrow a$. Thus, production $a \rightarrow a$ can be "overwritten", if necessary.

Definition 5.2. Let $G = \langle V, \omega, p \rangle$ be a pL-system, and suppose that μ is an arbitrary word over V . We will say that the word v is **directly derived** from μ and write $\mu \Rightarrow v$ iff there exists a sequence of productions $p^{(t)} = \eta_l^{(t)} < \eta^{(t)} > \eta_r^{(t)} \rightarrow \chi^{(t)}$ and a sequence of positions $s^{(t)}$ ($t = 0, \dots, k$; $k \leq m$) such that:

- $\mu = \eta^{(0)} \dots \eta^{(k)}$,
- $v = \chi^{(0)} \dots \chi^{(k)}$,

- $p^{(0)}$ is applicable to the string μ at position $s^{(0)} = 0$,
- $p^{(t+1)}$ is applicable to the string μ at position $s^{(t+1)} = s^{(t)} + \text{length}(\eta^{(t)})$ for all $t = 0, 1, \dots, k-1$.

Example. Consider a pL-system G with the following productions:

$$p_1 = X < XY \rightarrow YX$$

$$p_2 = X > X \rightarrow YXX$$

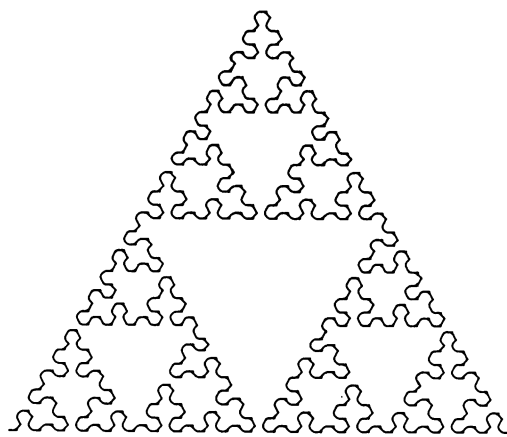
A word $\mu = \text{XXYXY}$ will be partitioned into strict predecessors as follows: $X \text{ XY } X \text{ Y}$. The corresponding successors are: $YXX \text{ YX } X \text{ Y}$. Thus, the word v directly derived from μ is equal to $YXXYXXY$.

The notion of the direct derivation is extended to the derivation of length n as in the case of OL-systems. Two curves generated by pL-systems are shown in Fig. 2.

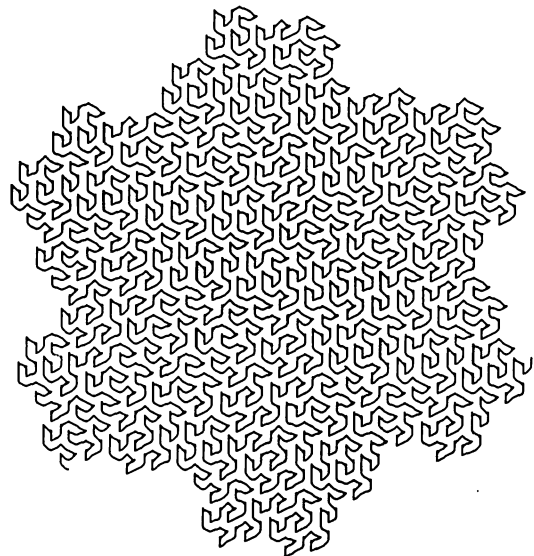
A useful extension of the set of commands interpreted by the turtle introduces two symbols "[" and "]" defined as follows:

- [Push current state of the turtle into a (pushdown) stack.
-] Pop a state from the stack, and make it the current state of the turtle. No line is drawn, although in general the position of the turtle is changed. If the stack is empty and no state can be popped, an error is reported and the string has no interpretation.

The above use of brackets is consistent with the original definition of L-systems by Lindenmayer [1968] where brackets were used to specify branches (of algae). This idea was also preserved in L-systems generating plants and trees for computer imagery purposes [e.g. Smith 1984]. Some plants and trees generated by OL-systems and pL-systems under the extended turtle interpretation are shown in Fig. 3.



a $n=6, d=4, \delta=60^\circ$
 YF
 $XF \rightarrow YF+XF+YF$
 $YF \rightarrow XF-YF-XF$



b $n=4, d=5, \delta=60^\circ$
 XF
 $XF \rightarrow XF+YF++YF-XF--XF XF-YF+$
 $YF \rightarrow -XF+YF YF++YF+XF--XF-YF$

Fig. 2. Examples of pictures generated by pL-systems under turtle interpretation. (a) The Sierpiński "arrowhead". (b) The Gosper space-filling curve. Both examples are taken from [Mandelbrot 1982].

Other graphical interpretation functions can also be considered. In the case of **chain interpretation** [Freeman 1961] letters **A**, **B**, **C** and **D** may be interpreted as commands moving the turtle to the left, up, to the right and down, respectively. These movements change position of the turtle by distance d , and are independent of the turtle's heading. Line segments connecting the old and the new position of the turtle are drawn. Under the chain interpretation, some interesting curves are generated by very simple L-

systems. For example, the dragon curve (Fig. 1e) is generated by a 0L-system with axiom **B** and productions $A \rightarrow AB$, $B \rightarrow CB$, $C \rightarrow CD$ and $D \rightarrow AD$. Further extensions of the interpretation functions are also possible. For example, lines within a pair of parentheses may define the boundary of a filled polygon (Fig. 4). The turtle can also be allowed to move in three dimensions. An L-system will then describe a three-dimensional object rather than a two-dimensional picture.

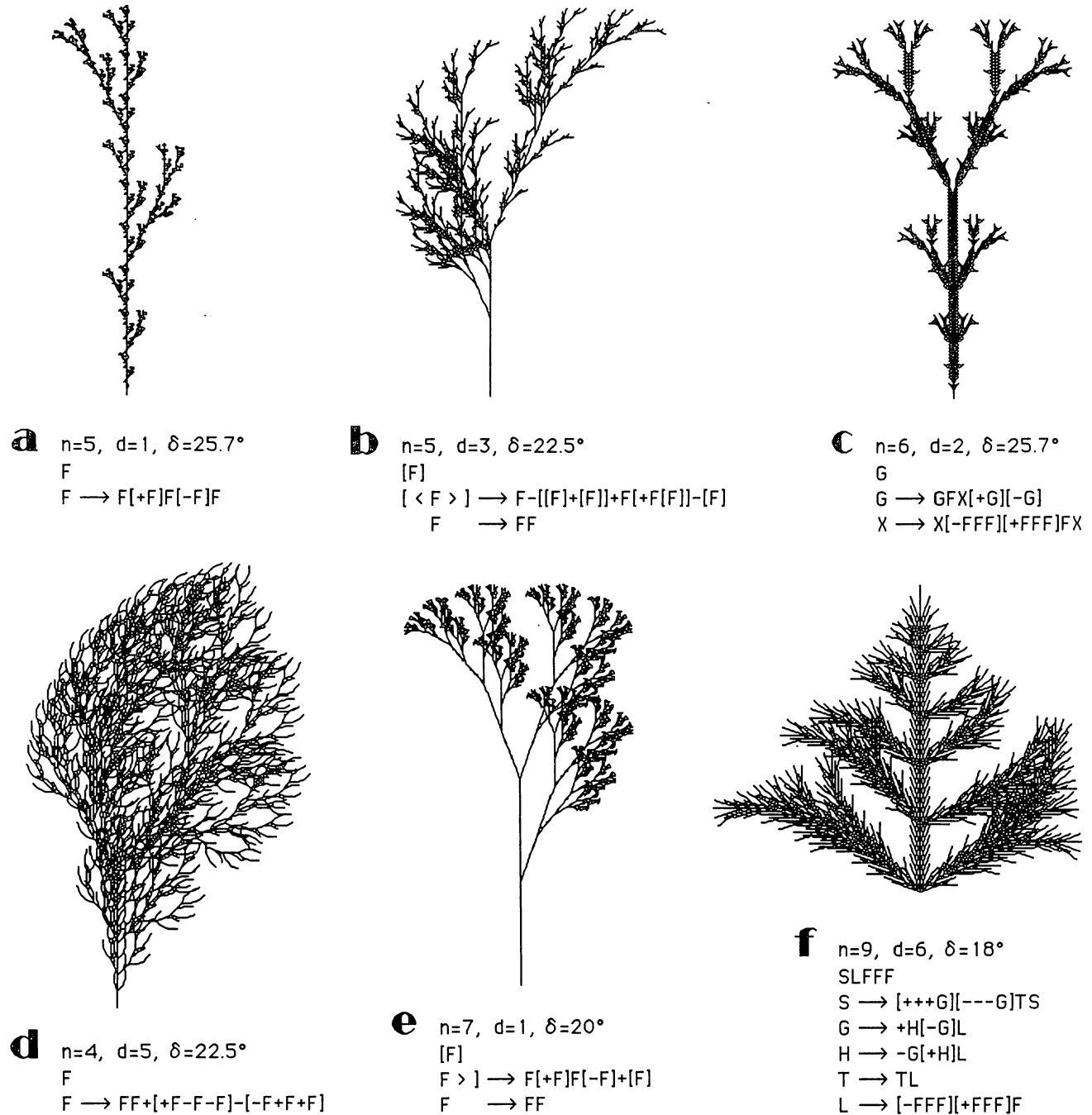


Fig. 3. Examples of plants and trees generated by OL-systems and pL-systems under the extended turtle interpretation.



$n=4$, $d=8$, $\delta=30^\circ$
 T
 $T \rightarrow R+[T]--[L]R[+L]-[T]++T$
 $R \rightarrow F[--L][+L]F$
 $L \rightarrow [(+FX-FX-FX+|+FX-FX-FX)]$
 $FX \rightarrow FX$
 $F \rightarrow FF$

Fig 4. Example of a plant generated by a pL-system. Parentheses are grouping edges which define boundaries of filled polygons.

6. CONCLUDING REMARKS.

This paper presents a technique for generating pictures consisting of two steps:

- A string of symbols is generated with an L-system,
- This string is interpreted graphically as a sequence of commands controlling a turtle.

The notion of the L-system is generalized to include productions in which predecessors may have an arbitrary length. The L-systems generalized this way are called pL-systems. Furthermore, the turtle is equipped with a pushdown stack which allows it to return to a previously marked position. The resulting mathematical model is used to define a large variety of fractals ranging from simple Koch curves popularized by Mandelbrot [1982], to classic space-filling curves, to relatively realistic-looking plants and trees. All these pictures are defined in a uniform and compact way. Consequently, L-systems can be used to define fractals in a similar way that equations are used to define analytic curves in the Cartesian coordinates. In other words, the description of the Sierpiński arrowhead by the productions $XF \rightarrow -YF+XF+YF-$, $YF \rightarrow +XF-YF-XF+$ may be as "natural" as the description of a circle by the equations $x = R \cos \theta$, $y = R \sin \theta$.

Many problems related to the graphical applications of L-systems remain open. One possible direction of future research consists of finding L-systems and interpretation functions suitable for generating visually attractive images. As is often the case with fractals, these images can be appealing either because of their abstract beauty, or because

of their similarity to real-life objects. Another research direction consists of exploring formal properties of L-systems related to picture generation. This is parallel to the study of graphical applications of Chomsky languages initiated by Maurer, Rozenberg and Welzl [1982]. Example problems are: Given two pL-systems and an interpretation function, are the resulting pictures congruent? Given a pL-system and an interpretation function, is the resulting line closed? Is it self-intersecting or tree-like? Are there any segments drawn more than once? Are they drawn infinitely many times (if the derivation length tends to infinity)? What is the function relating the diameter of the picture to the derivation length? Solutions to these problems are interesting not only from the theoretical point of view. They would also be useful when constructing pL-systems in order to generate pictures with given properties.

ACKNOWLEDGMENT.

The research described in this paper has been supported by grant number A0324 from the Natural Sciences and Engineering Research Council of Canada. This support is gratefully acknowledged.

REFERENCES.

- Abelson, H., and diSessa, A. A. [1982]: *Turtle geometry*. M.I.T. Press, Cambridge and London.
- Aono, M., and Kunii, T. L. [1984]: Botanical tree image generation. *IEEE Computer Graphics and Applications* 4, Nr. 5, pp. 10-34.
- Cole, A. J. [1983]: A note on space-filling curves. *Software - Practice and Experience* 13, pp. 1181-1189.
- Cole, A. J. [1985]: A note on Peano polygons and Gray codes. *Intl. Journal of Computer Mathematics* 18, pp. 3-13.
- Dacey, M. F. [1970]: The syntax of a triangle and some other figures. *Pattern Recognition* 2, pp. 11-31.
- Davis, C., and Knuth, D. E. [1970]: Number representations and dragon curves. *J. of Recreational Mathematics* 3, pp. 66-81 and 133-149.
- Feder, J. [1968]: Languages of encoded line patterns. *Information and Control* 13, pp. 230-244.
- Freeman H. [1961]: On encoding arbitrary geometric configurations. *IRE Trans. Electron. Comput.* 10, pp. 260-268.
- Fu, K. S. [1980]: Syntactic (linguistic) pattern recognition. In Fu K. S. (Ed.): *Digital pattern recognition*, Springer-Verlag, Berlin - Heidelberg - New York, pp. 95-134.
- Gips, J. [1975]: *Shape grammars and their uses; artificial perception, shape generation and computer aesthetics*. Birkhäuser-Verlag, Basel and Stuttgart.
- Goldschlager, L. M. [1981]: Short algorithms for space-filling curves. *Software - Practice and Experience* 11, p. 99.
- Herman, G. T., and Rozenberg, G. [1975]: *Developmental systems and languages*. North-Holland, Amsterdam and Oxford.
- Hilbert, D. [1891]: Ueber stetige Abbildung einer Linie auf ein Flächenstück. *Math. Annln.* 38, pp. 459-460.

- Kirsch, R. A. [1964]: Computer interpretation of English text and picture patterns. *IEEE Trans. Electron. Comput.* EC-13, pp. 363-376.
- Ledley, R. S. [1964]: High-speed automatic analysis of biomedical pictures. *Science* 146, No. 3641, pp. 216-223.
- Ledley, R. S., et al. [1965]: FIDAC: Film input to digital automatic computer and associated syntax-directed pattern-recognition programming system. In J. T. Tip-pet et al. (Eds.): *Optical and electro-optical information processing*, M.I.T. Press, Cambridge, pp. 591-613.
- Lindenmayer, A. [1968]: Mathematical models for cellular interaction in development, Parts I and II. *Journal of Theoretical Biology* 18, pp. 280-315.
- Lindenmayer, A., and Rozenberg, G. (Eds.) [1976]: *Automata, languages, development*. North-Holland, Amsterdam - New York - Oxford.
- Mandelbrot, B. B. [1982]: *The fractal geometry of nature*. W. H. Freeman, San Francisco.
- Maurer, H. A., Rozenberg, G., and Welzl, E. [1982]: Using string languages to describe picture languages. *Information and Control* 54, pp. 155-185.
- Narasimhan, R. [1962]: *A linguistic approach to pattern recognition*. Rep. 21, Digital Computer Laboratory, University of Illinois, Urbana.
- Narasimhan, R. [1966]: Syntax-directed interpretation of classes of pictures. *Comm. ACM* 9, pp. 166-173.
- Null, A. [1971]: Space-filling curves, or how to waste time with a plotter. *Software - Practice and Experience* 1, pp. 403-410.
- Papert, S. [1980]: *Mindstorms: children, computers, and powerful ideas*. Basic Books, New York.
- Peano, G. [1890]: Sur une courbe, qui remplit tout une aire plane. *Math. Annln.* 36, pp. 157-160.
- Pfaltz, J. L. [1972]: Web grammars and picture description. *Computer Graphics and Image Processing* 1, pp. 193-220.
- Rosenfeld, A., and Milgram, D. L. [1972]: Web automata and web grammars. *Machine Intelligence* 7, pp. 307-324.
- Salomaa, A. [1973]: *Formal languages*. Academic Press, New York and London.
- Shaw, A. C. [1969]: A formal picture description scheme as a basis for a picture processing system. *Information and Control* 14, pp. 9-52.
- Sierpiński, W. [1912]: Sur une nouvelle courbe qui remplit tout une aire plane. *Bull. Acad. Sci. Cracovie, Série A*, pp. 462-478.
- Siromoney, R., and Subramanian, K. G. [1983]: Space-filling curves and infinite graphs. In H. Ehrig, M. Nagl and G. Rozenberg (Eds.): *Graph grammars and their application to computer science*, Springer-Verlag, Berlin - Heidelberg - New York - Tokyo.
- Smith, A. R. [1984]: Plants, fractals, and formal languages. *Computer Graphics* 18, Nr. 3, pp. 1-10.
- Sudborough, I. H., and Welzl, E. [1985]: Complexity and decidability for chain code picture languages. *Theoretical Computer Science* 36, pp. 173-202.
- Stiny, G. [1975]: *Pictorial and formal aspects of shape and shape grammars*. Birkhäuser-Verlag, Basel and Stuttgart.
- Wirth, N. [1976]: *Algorithms + data structures = programs*. Prentice-Hall, Englewood Cliffs.
- Witten, I. H., and Wyvill, B. [1983]: On the generation and use of space-filling curves. *Software - Practice and Experience* 13, pp. 519-525.

FRACTALS, COMPUTERS AND DNA

Peter Oppenheimer

New York Institute of Technology, Old Westbury, NY 11568, USA

ABSTRACT

The goal of science is to understand why things are the way they are. By emulating the logic of nature, computer simulation programs capture the essence of natural objects, thereby serving as a tool of science. When these programs express this essence visually, they serve as an instrument of art as well.

This paper presents a fractal computer model of branching objects. This program generates pictures of simple orderly plants, complex gnarled trees, leaves, vein systems, as well as inorganic structures such as river deltas, snowflakes, etc. More than just a visual simulation, this program models the growth process by mimicking the logic of an organism's genetics. By manipulating the genetic parameters, one can modify the geometry of the object in realtime, using tree based graphics hardware. The random effects of the environment are taken into account, to produce greater diversity and realism.

The program provides a study in the structure of branching objects that is both scientific and artistic. The results suggest that organisms and computers deal with complexity in similar ways, and that the fractal nature of an organism has evolved as a critical means for the survival of the species.

RESUME

Le but de la science est de comprendre le pourquoi des choses. En imitant la logique de la Nature, les logiciels de simulations informatiques permettent cerner l'essence des objets naturels, et deviennent ainsi des outils scientifiques. Lorsque ces programmes de simulation expriment leur résultats de façon graphique, ils deviennent aussi des modes d'expression artistique.

Cette communication présente un modèle informatique pour la génération d'objets fractals arborescents. Le logiciel permet de générer des images de plantes de faible degré de complexité, des arbres nouveaux, des feuilles d'arbres, des systèmes ramifiés, mais aussi des systèmes du monde inerte comme des deltas de rivières, des

cristaux de neige, etc... Au delà de la simple modélisation visuelle, ce programme simule le processus de croissance de ces formes en imitant la logique génétique de ces organismes. En manipulant les divers paramètres de ce code génétique, on peut contrôler en Temps Réel la géométrie de l'objet, grâce à l'exploitation d'un matériel capable pour la gestion de structures de données en arbre. Les perturbations aléatoires rencontrées dans les formes de croissance réelles contribuent à renforcer le réalisme des images générées et augmentent la diversité des formes ainsi produites.

Le logiciel permet d'étudier des objets à structure arborescente aussi bien du point de vue scientifique que du point de vue artistique. Les résultats obtenus suggèrent que les organismes vivants et les ordinateurs présentent certaines analogies vis à vis de la gestion de structures de croissances complexes, et que la nature fractale de certains organismes a évolué vers un équilibre optimum permettant la survie de ces espèces.

1. INTRODUCTION

Benoit Mandelbrot recognized that the relationship between large scale structure and small scale detail is an important aspect of natural phenomenon. He gave the name *fractals* to objects that exhibit increasing amount of detail as one zooms in closer. [9][10] If the small scale detail resembles the large scale detail, the object is said to be self-similar.

The geometric notion of fractal self-similarity has become a paradigm for structure in the natural world. Nowhere is this principle more evident than in the world of botany. Recursive branching at many levels of scale, is the primary mechanism of growth in most plants. Analogously, recursive branching algorithms, are fundamental to computers. Many high performance processing engines specialize in tree data structures.

Computer generation of trees has been of interest for several years now. Examples of computer generated trees include

Benoit Mandelbrot (1977)(1982) [9],[10]
 Marshall, Wilson, Carlson (1980) [11]
 Yoichiro Kawaguchi (1982) [8]
 Geoff Gardiner (1984) [7]
 Aono, Kunii (1984) [2]
 Alvy Ray Smith, Bill Reeves (1984)(1985) [17][14]
 Jules Bloomenthal (1985) [4]
 Demko, Hodges, Naylor (1985) [6]

2. THE TREE MODEL

The tree model presented in this paper has the following features:

- A detailed parameterization of the geometric relationship between tree nodes.
- Real Time Design and Animation of tree images using high performance hardware.
- Application of Stochastic (random) Modeling to both topological and geometric parameters.
- Stochastic modeling of tree bark.
- High Resolution (2024 x 1980) Shaded 3d Renderings.

Here's how the model works:

This program implements a recursive tree model. Each tree generated satisfies the following recursive tree node definition:

```
tree :=
{
    Draw Branch Segment
    if (too small)
        Draw leaf
    else
    {
        # Continue to Branch
        {
            Transform Stem
            "tree"
        }
        repeat n times
        {
            Transform Branch
            "tree"
        }
    }
}
```

Paraphrased, a tree node is a branch with one or more tree nodes attached, transformed by a 3x3 linear transformation. Once the branches become small enough, the branching stops and a *leaf* is drawn. The trees are differentiated by the geometry of the transformations relating the node to the branches and the topology of the number of branches coming out of each node.

These branching attributes are controllable by a set of numerical parameters. Editing the parameters, changes the tree's appearance. The parameters include:

- The angle between the main stem and the branches
- The ratio of the main stem to the branches
- The rate at which the stem tapers
- The amount of helical twist in the branches
- The number of branches per stem segment

Figure 1 shows a simple example with mostly default parameters.

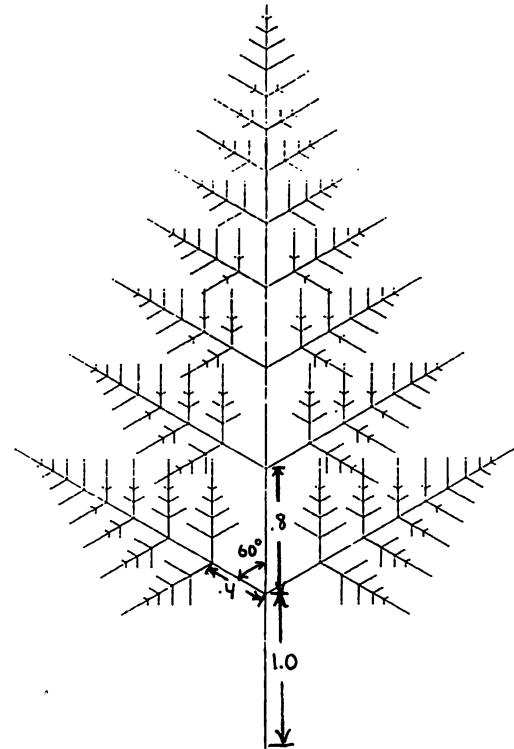


Figure 1

stem/stem ratio = .8
 branch/stem ratio = .4
 branching angle = 60°

3. RANDOM NUMBERS IN FRACTAL MODELING

If the parameters remain constant throughout the tree, one gets a very regular looking tree such as a fern. This tree is strictly self similar; that is, the small nodes of the tree are identical to the top level largest node of the tree.

If the parameters vary throughout the tree, one gets an irregular gnarled tree such as a juniper tree. In order to achieve this, each parameter is given a mean value and standard deviation. At each node of the tree, the

parameter value is regenerated by taking the mean value and adding a random perturbation, scaled by the standard deviation. The greater the standard deviation, the more random, irregular, and gnarled the tree. The resulting tree is *statistically* self-similar; not *strictly* self-similar.

There are several reasons for the stochastic approach. First, adding randomness to the model generates a more natural looking image. Large trees have an intrinsic irregularity (caused in part by turbulent environmental effects). Random perturbations in the model reflect this irregularity. Second, random perturbations reflect the diversity in nature. A single set of tree model parameters can generate a whole forest of trees, each slightly different. This increased database amplification is one of the hallmark features of fractal techniques.

4. MODELING STEM SHAPE

By stripping all of the branches one is left with just a stem. By varying the transformation between stem segments, one derives the class of *spirals* and *helixes* and their random perturbations. These shapes appear in all forms of growth, organic and inorganic -- from the inner ear, sea shells, and plant sprouts, to spiral galaxies. *Spirals* and *helixes* are in some sense degenerate self similar sets. They are the atomic units that make up the fractal trees.

Figure 2 shows 4 typical *stem* shapes.

- a) *cylinder*: the transformation is a translation and a scale.
- b) *spiral*: one performs a rotation perpendicular to the stem axis, in addition to a scale and translation.
- c) *helix*: one performs an additional rotation along the stem axis.
- d) *squiggle* By randomly changing the transformation from segment to segment, case c) becomes case d).

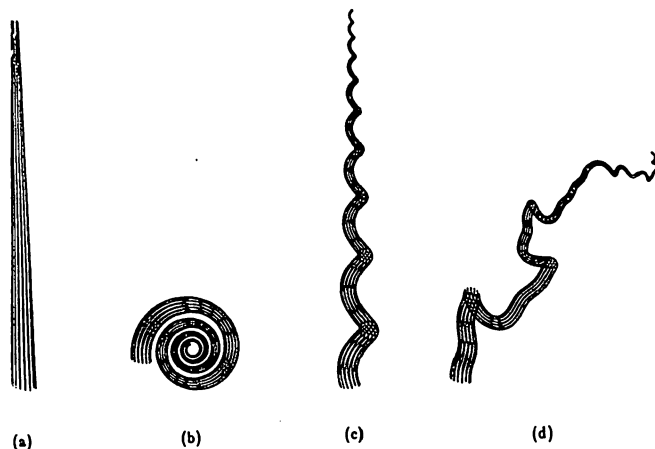


Figure 2

Spirals & Helixes

Branches are simply stem shapes attached to the main stem and each other.

5. RENDERING THE FRACTAL

A variety of geometric elements can be used to render the branch segments. The simplest primitive is one single vector line per tree node. Antialiased vector lines with variable thickness allow one to taper the branches towards the tip. This method is satisfactory for leaves, ferns and other simple plants, for small scale detail in complex scenes, or for more abstract stylized images. Varying the vector color provides depth cuing and shading, and can also be used to render blossoms or foliage. Antialiased vectors are similar to particle systems used by Bill Reeves in his forest images. [14]

The thicker branches of a tree require a shaded 3d primitive. Bump mapped polygonal prisms are used to flesh out the trees in 3d. The program makes sure that the polygons join continuously along each limb. The branches emanating from a limb simply interpenetrate the limb. For a more curvilinear limb shape, one can link several prisms together between branch points.

Shaded polygon limbs are far more expensive than antialiased vectors. Since the number of branches increases exponentially with branching depth, one can spend most of an eternity rendering sub pixel limb tips, where bark texture and shading aren't visible anyway. In addition to being faster, sub pixel vectors are easier to antialias than polygons on our available rendering package. So for complex trees with a high level of branching detail, polygonal tubes were used for the large scale details, changing over to vectors for the small stuff. One can notice the artifacts of this technique. Overall, however, the eye ignores this inconsistency if the cutover level is deep enough. Thinner branches require fewer polygons around the circumference; in fact triangular tubes will do for the smallest branches.

6. BARK

Sawtooth waves modulated by Brownian fractal noise are the source for the simulated bark texture. bark is generated by adding fractal noise to a ramp, then passing the result through a sawtooth function. A close up view of the bark would look like the ridges of a fractal mountain range. By adding the noise before the sawtooth function, the crests of the sawtooth ridges become wiggly.

7. REAL TIME FRACTAL GENERATION

Complex tree images can take 2 hours or more to render on a VAX 780. Editing tree parameters at this rate is not very effective. Near real time feed back is needed to allow one to freely explore the parameter space, and design the desired tree. Since vertex transformation cost is high for a complex fractal tree, hardware optimized for linear transformations was used for the real time editor. The Evans and Sutherland MultiPictureSystem generates vector drawings of complex 3d display lists in near real time. The display lists on the MPS look a lot like our tree nodes: primitive elements,

transformed by linear transformations, and linked by pointers to other nodes. For a strictly self similar tree, the transformation is constant, therefore the entire display list can share a single matrix. To modify the tree one only has to change this one matrix, rather than an entire display list. This makes updating the tree display list very fast. For non-strictly self similar trees, the transformations are not the same. However a lookup table of less than a dozen transformations, is adequate to provide the necessary randomness. [17]

Figure 3 illustrates the logic of the display list.

The left side of the display list contains the topological description of the tree. Each node contains pointers to offspring nodes plus a pointer to the transformation matrix which relates that node to its parent node. This part of the display list is purely topological: it contains no geometric data. The geometric information is contained in the small list of transformations. matrices on the right. To edit a tree, one can create an arbitrarily large topology list once and then rapidly manipulate only the small geometry list. Alvy Ray Smith in his research on *graftals*, recognized the separation of the topological and geometric aspects of trees. He calls these components the graph, and the interpretation respectively. His work deals primarily with specification of the tree topology, ignoring interpretation for the most part [17]. The paper presented here emphasizes the geometric interpretation. The thesis of this paper is that the key to realistically modeling the diversity of trees lies in controlling the geometric interpretation. Many different topologies were used in this project. But by varying the geometric interpretation of a *single* topology, one could still generate a wide variety of trees each with its own distinct taxonomic identity.

The real time generation of fractal trees has been packaged as an interactive editing system. This multi-window system allows one to edit the tree parameters, (both geometric and topological) via graphically

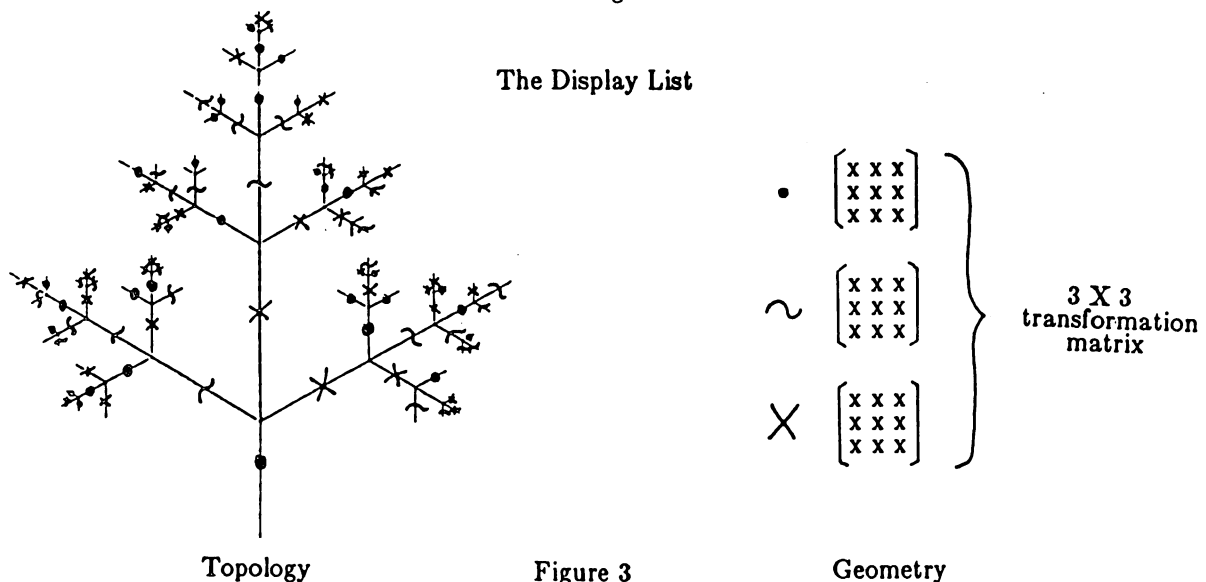
displayed sliders. A vector image of the tree responds in real time. To see all the parameters change at once one performs keyframe interpolation of the parameters. Each tree parameterization is written to a keyframe file. A cubic spline program, interpolates these parameters to create the inbetween frames. In the resulting animation, the tree metamorphoses from key to key. A simple *tree growth* animation is achieved by interpolating the trunk width parameter, and the recursion size cutoff parameter. Modifying additional parameters makes the growth more complex and natural looking. For example, many plants uncurl as they grow. A metamorphosis animation is achieved by interpolating parameters from different tree species. Growing and metamorphosing tree animations appear *The Palladium* animation produced at NYIT. [1]

The above method for real time fractal animation on the Evans and Sutherland Multi-Picture System using vectors is currently being transported to the CGL/Trillium real time smooth shaded polygon rendering system.

8. FRACTALS, COMPUTERS, AND DNA

The economic advantage of this program is that a highly complex structure is generated from a simple concise kernel of data which is easy to produce. (Such large database amplification is a primary advantage of fractal techniques in general.) How does the representation of complexity by computers compare with complex expression in nature itself?

Presumably, complexity in nature has evolved because it can bestow benefits on an organism. But, as with computers, complexity must not be a burden. An organism must be simple to build, simple to describe. After all, a mere picogram of DNA serves as the blueprint for animals weighing tons. Genetic economy demands that intricate structures be summarized simply. This struggle to simplify genetic requirements, determines the geometric structure of the plant. Form follows genetic economics.



This suggests that a natural explanation as to why self-similarity abounds in the natural world: evolution has resolved the tension between complexity and simplicity in the same way that computer scientists have -- with recursive fractal algorithms. If fractal techniques help the computer resolve the demands of database amplification, then presumably organisms can benefit as well. For genes and computers alike, self-similarity is the key to thrifty use of data.

The parameters of the tree program are numerical counterparts to the DNA code that describes a tree's branching characteristics. The logic of the gene is mimicked although the mechanism is different. The early stages of the model contained only 3 changeable parameters. The resulting images were of very simple fern like plants. New species were generated by controlling the parameter values, rerolling the dice of mutation and then selecting the forms that would be allowed to proliferate. As the model became more complex with the inclusion of more parameters, the program created images of more genetically complex trees such as cherry trees, higher on the evolutionary scale. Whereas natural selection of organisms is based on survival value, this aesthetic selection is based upon resemblance to the forms of nature.

The actual computer program did not take very long to develop, just as it did not take long for plants to develop the ability to branch. Expanding the parameter space, and creating a diverse database, however, required turning the dials throughout all four seasons. This parameter space represents a more evolved instance of previous tree parameterizations.

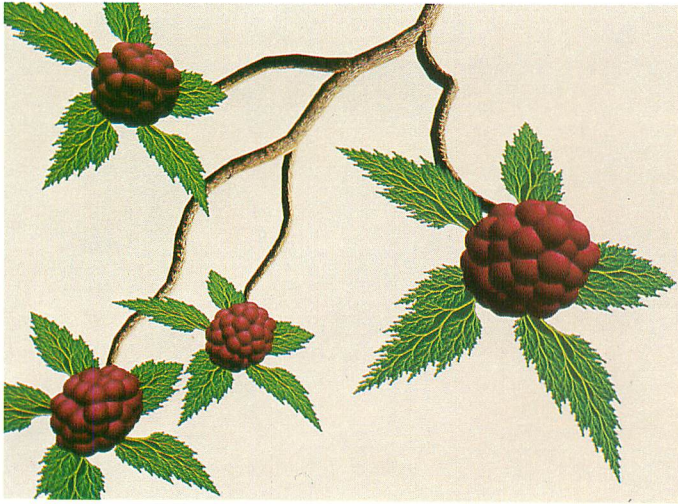
9. CONCLUSION

Of course any scientific model is simply an attempted translation of nature into some quantifiable form. The success of the model is measured by some qualitatively predictive result. In experimental science, the success of a theory is measured by the degree to which the predicative model matches experimental data. Computer graphics now provides another style of predictive modeling. The success of a computer simulation is reflected in how well the image resembles the object being modeled. If the picture looks like a cherry tree, this suggests that the model is "correct." If one can model a complex object through simple rules, one has mastered the complexity. What appeared to be complex, proves to be primitive in the end. And the proof (although subjective) is in the picture.

REFERENCES

- [1] Allen, R., Oppenheimer, P., *The Palladium* (Video), New York Institute of Technology, 1985
- [2] Aono, M., Kunii, T.L., *Botanical Tree Image Generation*, IEEE Computer Graphics and Applications, Vol. 4, No. 5, May 1982

- [3] Bentley, W.A., Humphreys, W.J., *Snow Crystals*, Dover Publications Inc., New York, 1962 (Originally McGraw Hill, 1931)
- [4] Bloomenthal, J., *Modeling the Mighty Maple*, Computer Graphics, Vol. 19, No. 3, July 1985.
- [5] Cole, V.C., *The Artistic Anatomy of Trees*, Dover Publications Inc., New York, 1965 (Originally Seeley Service & Co, London, 1915)
- [6] Demko, S., Hodges, L., Naylor, B., *Construction of Fractal Objects with Iterated Function Systems*, Computer Graphics, Vol. 19, No. 3, July 1985.
- [7] Gardiner, G., *Simulation of Natural Scenes Using Textured Quadric Surfaces*, Computer Graphics, Vol. 18, No. 3, July 1984.
- [8] Kawaguchi, Y., *A Morphological Study of the Form of Nature*, Computer Graphics, Vol. 16, No. 3, July 1982.
- [9] Mandelbrot, B., *Fractals: Form, Chance and Dimension*, W.H. Freeman and Co., San Francisco, 1977.
- [10] Mandelbrot, B., *The Fractal Geometry of Nature*, W.H. Freeman and Co., San Francisco, 1982.
- [11] Marshall, R., Wilson, R., Carlson, W., *Procedural Models for Generating Three-Dimensional Terrain*, Computer Graphics, Vol. 14, No. 3, July 1980.
- [12] Oppenheimer, P. *Constructing an Atlas of Self Similar Sets* (thesis) Princeton University, 1979.
- [13] Oppenheimer, P. *The Genesis Algorithm*, The Sciences, Vol 25, No 5., 1985.
- [14] Reeves, W., *Particle Systems--A Technique for Modeling a Class of Fuzzy Objects*, Computer Graphics, Vol. 17, No. 3, July 1983.
- [15] Reynolds, C., *Arch Fractal*, Computer Graphics (Front Cover), Vol. 15, No. 3, August 1981.
- [16] Serafini, L., *Codez Seraphinianus*, Abbeville, New York, 1983.
- [17] Smith, A.R., *Plants, Fractals, and Formal Languages*, Computer Graphics, Vol. 18, No. 3, July 1984.
- [18] Stevens, P.S., *Patterns in Nature*, Little, Brown, and Co. Boston, 1974.



Fallen Leaf - An exaggerated image of vein branching in a fall leaf. The external boundary shape is the limit of growth of the internal veins. (512x480)



Raspberry Garden at Kyoto - The leaves and branches are generated by the fractal branching program. Berries are based on symmetry models by Hareh Lalvani, with added random perturbations. Non-self similar features such as berries, require genetic specialization. (1024x960)



Blossomtime - The bark of this cherry tree is made with bump mapped polygonal tubes. The blossoms are colored vectors (particle systems) Instead of modeling blossoms, one can simply *dip* the branches in pink paint. (1024x960)

Views II - By randomly perturbing the branching parameters, one generates a more naturalistic gnarled tree. (2048x1920)

A Knowledge-Based Approach To Computer Vision Systems

Martin D. Levine
Wade Hong

Computer Vision and Robotics Laboratory
Department of Electrical Engineering
McGill University
Montréal, Québec, Canada

Abstract

In designing and constructing computer vision systems, many crucial issues need to be addressed. Foremost of these are the control and organization of the visual information processing tasks involved, and the representation and usage of both knowledge and data. As computer vision systems have evolved, growing in complexity and size, these issues have become increasingly important to their overall success. In this paper, a recent and increasingly popular approach to image understanding, the knowledge-based system, is presented as a framework in which to deal with these issues. The engineering of a computer vision system as a knowledge-based system and these issues, in the context of our evolving system is discussed.

Résumé

Lors de la conception et de la mise en oeuvre d'un système de vision par ordinateur, plusieurs questions critiques doivent être considérées. Principalement, il s'agit du contrôle et de l'organisation des tâches de traitement d'information visuelle ainsi que de la représentation et de l'usage des données et des connaissances. Parce que les systèmes de vision par ordinateur ont évolué en grandeur et en complexité, leur succès dépend de plus en plus de ces questions. Dans cet article, une approche nouvelle et de plus en plus populaire à la compréhension d'images, le système basé sur les connaissances, est présentée en tant que cadre de travail pour traiter ces questions. La réalisation d'un système de vision par ordinateur par le biais d'un système basé sur les connaissances ainsi que ces questions sont traitées dans le contexte de notre système en évolution.

1. Introduction

A visual technology capable of replicating human vision is the ultimate achievement for computer vision. To be able to accomplish such a feat would require a far superior understanding of the functioning of the human

visual system. Moreover, this would require the embodiment of intelligence in a machine. Undaunted by these severe limitations in understanding, *computer vision* has developed over the past twenty-five years in a somewhat ad hoc fashion. The growth of this infant technology in conjunction with its maternal science of artificial intelligence has led to the emergence of computer vision systems. Albeit they are far from being general vision systems⁺ they are at present the best and only available artificial approximation.

The earliest computer vision system, pioneered in the mid 1960's by Roberts [Roberts65], was capable of analyzing simple polyhedral scenes and matching the located polyhedra to stored models. Since then, computer vision systems have attained greater complexity due to the increasingly complex scenes being analyzed, as witnessed in the prominent systems of today. (See [Binford82] and [Shapiro83] for surveys on some of these systems.) In association with this increase in complexity, the control and organization of these systems have evolved from simple sequential bottom-up or top-down mechanisms into complex structures involving many levels of cooperative processes, as the amount of knowledge required to reason about the analysis increases. As these complex visual information processing systems become more ambitious, it is clearly evident that the organization and control aspects will also become increasingly more significant to their overall success.

Control of vision systems have tended to be heavily embedded within the organization of the visual processes. Such procedural methods are reliable and fast, but are very rigid in that they are application specific. Subject to variations in the goal description or the task domain, the appropriate alterations to the procedural knowledge may become a major task. Also, if the images to be analyzed consist of complex structures and great intra-class variations, a sequence of analysis cannot be reliably predetermined. Thus the analysis is necessarily data-driven, implying the need for a flexible and adaptive control struc-

⁺By general it is meant in the same sense as the human visual system, capable of multiple objectives in a dynamic, unconstrained and complex visual environment.

ture.

This paper presents a recent and increasingly popular approach to the organization of a computer vision system, permitting a greater degree of control flexibility and subsequently, functional generality. The paradigm presented is that of a knowledge-based system.

2. The Knowledge-Based Approach

A significant result in the first twenty years of artificial intelligence research is the fact that the principal requirement for intelligence is *knowledge*. By the mid-1970's AI began shifting from a power-based strategy towards a knowledge-based approach in an attempt to achieve intelligence. The power strategy looked towards a generalized increase in computational power in resolving the problems that the current techniques faced, whereas the knowledge strategy viewed progress being achieved from better ways of recognizing, representing and utilizing diverse and specific forms of knowledge. The fundamental problem of understanding intelligence is no longer the identification of power-based techniques, but rather a question of how to represent vast amounts of knowledge in a manner which permits their effective use and interaction.

A powerful tool that has emerged from this shift of focus in AI is the knowledge-based system which is a problem solving system that applies knowledge about a specific domain to solve practical problems [Sowa84]. A class of knowledge-based systems known as expert systems has recently received much attention [Waterman, Hayes-Roth&Lenat83].

Knowledge-based systems have either adopted or developed programming styles where there exists a clear distinction between knowledge and its use (for an introduction to and survey of a few of existing tools, see [Waterman, Hayes-Roth&Lenat83], pp. 169-215). This separation of control flow from its knowledge permits modular extensions to a system's capabilities. The knowledge engineering tools that have emerged employ principles of knowledge representation and a related inference mechanism for bringing knowledge to bear on a problem. Knowledge about the problem domain and self-knowledge are stored in a knowledge base using a representational framework. Current representational frameworks include rule-based, frame-based and logic-based schemes [Buchanan&Duda83]. Facts or data about the particular problem and processing are stored in a global database. The system retrieves pertinent knowledge to the problem and utilizes symbolic reasoning to make inferences about the facts in the global database to solve the problem at hand.

Although one of the first domains of research in AI to incorporate knowledge was computer vision, the extent of improvement in this application has been slow and limited. The application of knowledge has been restricted to domain specific knowledge of the scene analyzed in

model-based vision. However, the use of world knowledge has been weak [Binford82]. There is now interest in the computer vision community to apply knowledge-based system techniques to improve this level of processing [Matsuyama 84, Nagao 82].

As complex and large as current computer vision systems are, they are very limited in their abilities [Binford82, Matsuyama84]. Much effort, of late has been directed towards improving and understanding specific vision tasks, particularly, in low level vision [Brady82]. A major emphasis in this work has been focused on the use of physical knowledge - knowledge about the physical world and the laws that govern it. Shape from shading and stereo vision, for example, use knowledge about the imaging process to recover 3D shape from projected 2D image features. More recently, another level of knowledge has been introduced in computer vision systems, perceptual knowledge - knowledge used to group image features into aggregates. The basis for this knowledge comes from Gestalt laws of visual grouping [Zucker, Rosenfeld&Davis75]. Such knowledge has been successfully applied in refining low level segmentations [Nazif83] and forming perceptual groupings from 2D image features as the basis for 3D object recognition [Lowe84].

Apart from the need to improve every facet of the image analysis process, there is also a need to increase the overall intelligence of these image understanding systems [Rosenfeld82, Matsuyama84]. The capacity of intelligence implies the ability to reason about the image analysis and the scene. Rosenfeld identifies a lack of a general theory of control in image analysis, i.e. there exists no general principles describing how vision processes should interact in performing a particular task. He also identifies a lack of a general theory of how to combine evidence from multiple sources of information available in performing a particular task. Such general purpose knowledge is imperative if hopes of achieving a general vision system are to be satisfied.

To achieve functional generality, a computer vision system must be capable of performing a variety of tasks. Upon specification of a particular task, the system must be able to determine the necessary processing modules, parameters and control strategy for performing the task. Given the requirement of being able to analyze a wide variety of complex images, this cannot be rigidly specified a priori. The system should possess the ability to evaluate its performance at various stages of processing and be capable of adaptively improving it (whether it be by modifying parameters, modifying the control flow, integrating information, augmenting processes, or other mechanisms). Thus, it is necessary that the image being analyzed and its many abstractions dictate the processing flow and consequently, how the vision processes should interact. The control of the image analysis is therefore necessarily data-driven. This type of flexible control is easily realizable in a knowledge-based methodology.

Ultimately, computer vision must address the im-

portant issue of integration of evidence from multiple sources, especially in view of the increased sophistication in applications and the need for improved performance. This is especially desirable since descriptions produced by computer vision techniques are incomplete and often imprecise, stemming from the inherent ambiguities that arise in an image. For example, consider the problem of image segmentation where partitions may be obtained from several measurable or extractable properties such as colour, luminance, texture or edges. In typical computer vision applications the "best" technique for segmenting the image, based on a single property, is often used to build an intermediate representation[‡] for the higher level processes. This "best" technique is often arrived at by trying a set of techniques and deciding on the best. However, it is necessary in a general system, where the "best" technique is not definable, to have a larger number of techniques available, and in some way be capable of integrating the results of these techniques into a "best" possible usable intermediate representation. Integration of this nature can be viewed as a refinement process which operates on local extracted features. Nazif [Nazif83] has demonstrated the refinement of low level segmentations using a rule-based mechanism to represent processing knowledge for integrating information from a line-based and a region-based segmentation. Note that the integration of information can also be useful in the refinement of the interpretation or recognition processes.

Given the importance of knowledge in image analysis, the engineering of a computer vision system as a knowledge-based system is very appealing. However, to have successful systems, the knowledge levels (physical, perceptual, domain and processing) must be further enhanced and the use of this knowledge be more effectively applied. Also, an appropriate knowledge engineering tool for vision applications must be formalized.

3. Our System

The aim of our system is to build a general purpose tool for experimenting with various approaches to image analysis. Constructing the system as a knowledge-based system permits us the flexibility to do so. In such a system where there is a distinct separation between its knowledge and the mechanisms that apply it, the task domain or its goals may be changed easily and as the system evolves, the modular extensibility of its capabilities by simply augmenting its knowledge is attractive. Equipped with a large set of visual processing algorithms and modules, by setting up the task domain and selecting the appropriate analysis strategy, this computer vision

system can attain a greater degree of functional generality and utility. Also, due to its data-driven nature, this system can be attentive to the processing requirements as dictated by the image, demonstrating the capacity of dynamic control [Levine&Nazif85b].

The basic computer vision system is identified as consisting of two major processing modules performing the low level or early processing and the high level or cognitive processing. Low level processing is concerned with extracting image features and structures to build an intermediate representation. The principal task of the high level process is to match object models with structures described in the intermediate representation. Achieving object recognition or scene interpretation is the product of both of these levels of processing. A meta supervisor coordinates the interaction and flow of information and processing between both processes. This simple organization is depicted in Figure 1. We follow the doctrine of separating the domain independent knowledge from the domain dependent knowledge in this form of dichotomy.

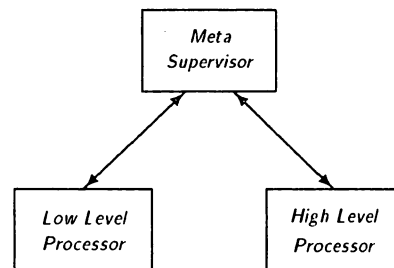


Figure 1 Basic System Structure

The organization of this system is presented in this fashion to express flexibility and generality which is permitted by the knowledge-based system paradigm. Although the interaction between the low level processor and the high level processor may be simply a one pass sequential flow or be governed by a hypothesis-verification paradigm, this arrangement permits either explanation. The point is not to mask the control structure but to emphasize that a knowledge-based approach permits greater flexibility in control. Changes in control strategy require only alterations in the meta control knowledge embedded in the meta supervisor or its usage as opposed to major reorganization necessary in a more conventional procedural control structure. Conceptually, the low level and high level processors and their respective subtasks are viewed in the same manner. For example, the low level processor has its meta supervisor controlling its subprocesses and similarly these subprocesses have supervisors controlling their respective subprocesses. Each of these respective processes are themselves self-contained knowledge-based systems. Organizing the system in this fashion suggests a natural pyramid or tree hierarchy for the control of the entire system.

[‡] Intermediate representation is the general term used to describe the representations produced at various stages of processing between the signal (image) and the semantic (scene) levels. For our purposes by intermediate representation, we mean the principal representation that is used by the interpretation (high level) process

4. Our Current Work

A system of the nature described above is currently evolving at the Computer Vision and Robotics Laboratory at McGill University. The knowledge representation framework chosen for the system implementation was a rule-based methodology and OPS5, a production system language [Forgy81, Dill&Hong84] was selected as the knowledge engineering tool. This latter choice was based primarily on availability.

A low level processor based on Nazif's low level segmentation expert [Nazif83] has been implemented and is currently being tested. Extensions to the capabilities of this system are currently being implemented. Work will be initiated soon on the high level processor.

The low level processor possesses the ability of non-purposive segmentation. A final partitioning of an image is obtained from the integration of initial region- and line-based segmentations. This integration is facilitated by the three knowledge sources which comprise the segmentation module: the line, region and area analyzers (see Figure 2). Each of these analyzers consists of rules which reason about the entities extracted from the image, i.e. lines, regions and areas of attention. These heuristics are domain independent, being based on the principles of visual grouping [Nazif83, Zucker, Rosenfeld&Davis75]. As well as the need for these heuristics to achieve the segmentation, some knowledge about how to apply them is also required. Hence the control problem.

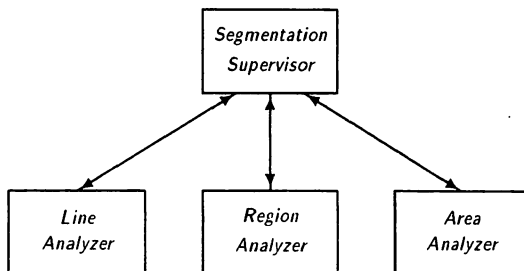


Figure 2 The Segmentation Module

Control is effected by dynamically setting strategies for the processing of areas, regions and lines. The selection of the strategies is based on a fuzzy concept of a region's or line's "need for further processing". A measure of this fuzzy notion is discernable from a set of performance parameters [Levine&Nazif85a, Nazif83] reflecting the quality of the segmentation at that instant in processing. Such a control strategy is very appealing in that it attends to the needs of the current segmentation and also by nature is domain independent.

The resulting intermediate description obtained from this segmentation module is a region-based representation of the image. However, the low level processor that is envisioned would combine many functional modules to

provide a rich intermediate representation of the scene, of which the segmentation module is one (see Figure 3). A second module now being implemented, which transcends the picture domain, is concerned with the extraction of scene domain cues. Such three-dimensional cues as occlusion, cast shadows, and skewness, extractable from the image contour, gives rise to some depth and orientation information. Exploiting this information, the shapes of objects may be inferred. This would yield an object-based segmentation of the scene. Similar to the segmentation module, the resulting partition of the scene would be obtained from the integration of the refined, region-based segmentation and this initial object-based segmentation. With the addition of other modules (perhaps a segmentation based on texture or a surface recovery module based on laser vision), the required integration would certainly be of greater complexity.

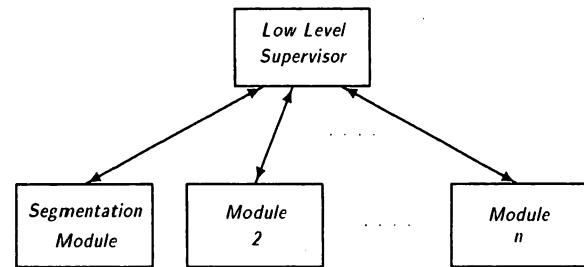


Figure 3 The Low Level Processor

The described low level processor, a general purpose subsystem by design, is oblivious to the task domain. It is in the high level processor where interaction with world knowledge is a necessity to achieve recognition or interpretation tasks. To accomplish this, the high level processor must possess the ability of matching object models to the intermediate representation supplied by the low level processor. More specifically, it must be able to resolve ambiguity (which is inherent in both the image data and world knowledge) and to identify instances of the object models by examining the consistency among local image features.

Some common paradigms that have been employed in image analysis include constraint propagation, template matching and hypothesis-verification [Matsuyama84, Binford82]. In these methods, initially some match or inference is made of image features to object models. Then these initial inferences are verified for local consistency whether in a sequential manner as is the case for template matching and hypothesis-verification, or in parallel for constraint propagation. Local consistency at some level is sufficient for object recognition tasks, but for interpretation, the inferences must be propagated to attain global consistency. These paradigms may be viewed as consisting of two characteristic mechanisms, one to make the initial inferences or matches and the other to propagate them (see Figure 4).

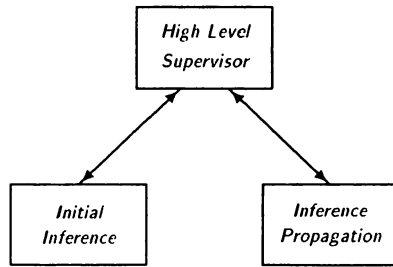


Figure 4 The High Level Processor

The objective of this high level processor is to achieve a scene description or object recognition given an object-based intermediate representation. But because the high level process is inherently limited by the quality of low level segmentations, ambiguity may not be easily resolved. Therefore, the high level process should have the ability to integrate evidence from other intermediate representations (region-based, line-based, etc.) in the inference forming and propagation processes. As a final recourse in the face of unresolvable ambiguity, the high level process should be able to request that the low level process either further refine or re-construct, a part or the whole of the intermediate description.

Work is now being initiated on the development of such a high level processor.

5. Discussion

Though the construction of a computer vision system as a knowledge-based system is very attractive, problems do however present themselves. They stem from the limitations and deficiencies of the representational framework, the knowledge engineering tool, its data-driven nature and knowledge itself. These shortcomings are not unique to this application, they are apparent in knowledge-based systems in general.

A major part of the effort in building a knowledge-based system is the identification and acquisition of pertinent knowledge applicable to the problem. Such knowledge is limited in its scope, incomplete and inexact, because we lack complete laws and theories about the problem. This is representative of the various knowledge levels (physical, perceptual, domain and processing) present in computer vision systems. Often the knowledge is ill-specified because it is not clear what exactly is known about the problem or how to apply it. To improve the performance of knowledge-based visual information processing systems a greater amount of knowledge must be identified and applied to the problem. Unlike the domain of expert systems, where there exist experts from which knowledge is accessible through interaction, knowledge useful to computer vision systems must be determined from the slow process of understanding human vision.

Control flow in a computer vision system such as ours is governed by the data, but this data is often unreliable

and incomplete. As a consequence, such a system could easily run astray. Coupled with ill-specified knowledge, the possibility is even greater. To cope with this problem, either the integrity of the data must be substantiated in some manner, by for example, incorporating redundancy (confirmation or combination of evidence from multiple sources) or the ability to reason with uncertainty must be established.

Although OPS5 is a general purpose production system programming language, our experience has shown that as a tool for constructing computer vision systems it suffers from several deficiencies. The principal one is that is inadequate for representing the diverse knowledge and data that must be embodied. The predominant nature of knowledge that must be encapsulated, especially at the low level is procedural; that is, it prescribes a set of operations. However, OPS5 does not facilitate procedural mechanisms nor complex computations on the right hand side of a rule. To capture a "chunk" of knowledge often requires the chaining of a set of productions. As well, there exist no generic control mechanisms that permit the accessing of a set of data in an orderly fashion, that is, the application of a rule (or a set of rules) sequentially on a set of data. Nevertheless, it is actually possible to accomplish this, but it requires the construction of specific control rules and the generation of control state data to ensure the proper processing flow. Finally, the data representation capabilities of OPS5 do not facilitate the representation of the lowest forms of visual data. There are no data structures for maintaining images, nor are there constructs to manipulate them.

These inadequacies and others using this knowledge engineering tool, though not insurmountable, suggest that perhaps some of our future work should be directed towards developing a more suitable knowledge engineering tool for constructing knowledge-based computer vision systems. An adequate tool would make the system more efficient and manageable. However, the specification of such a tool would require one to first identify the requirements necessary for building a knowledge-based computer vision system.

The rule-based methodology is a very general and flexible framework for representing knowledge and data, as is evident by its prevalent use in expert systems, covering a wide scope of problem domains. Even so, it is found to be not entirely adequate for our purposes. Subject to the nature of certain representations and processing requirements in our system, our experiences with OPS5 as discussed above, have shown that a classic pure production system model has its deficiencies. This suggests that a purely rule-based representational framework is not appropriate. A blend of the rule-based model and the imperative model would be more suitable.

The work that we have described here is only in its formative stages. Though we cannot yet conceive of all the many problems that will face us, we are however beginning to understand some of the major issues in-

volved in attempting to build such a massive system. This knowledge will become invaluable in the future evolution of this system.

Acknowledgements

This work was partially supported by the Natural Sciences and Engineering Research Council and the Ministry of Education of the Province of Québec. Martin D. Levine would like to thank the Canadian Institute for Advanced Research for its support. The authors would also like to thank Kamal K. Gupta for his constructive comments.

References

- [Binford82] Binford, T.O., *Survey of Model-Based Image Analysis Systems*, The International Journal of Robotics Research, Vol. 1, No. 1, MIT Press, Cambridge, Massachusetts, 1982.
- [Brady82] Brady, M., *Computational Approaches to Image Understanding*, ACM Computing Surveys, Vol. 14, No. 1, March 1982.
- [Buchanan & Duda 83] Buchanan, B.G., and Duda, R.O., *Principles of Rule-Based Expert Systems*, Advances In Computers 22, Academic Press, New York, 1983, 163-216.
- [Dill&Hong84] Dill, A.R., and Hong, W., *Everything You Wanted To Know About OPS5 But Were Afraid To Ask (A Practical Guide To Using OPS5 at CVaRL)*, Technical Report 84-5R, Computer Vision and Robotics Laboratory, Department Of Electrical Engineering, McGill University, Montréal, August, 1984.
- [Forgy81] Forgy, C.L., *OPS5 User's Manual*, Technical Report CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1981.
- [Havens&Mackworth83] Havens, W.S. and Mackworth, A.K., *Representing Knowledge of the Visual World*, IEEE Computer, Vol. 16, No. 10, October, 1983, 90-98.
- [Levine&Nazif85a] Levine, M.D. and Nazif, A.M., *Measurement of Computer Generated Image Segmentations*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 2, March, 1985, 155-164.
- [Levine&Nazif85b] Levine, M.D. and Nazif, A.M., *Rule-Based Image Segmentation: A Dynamic Control Strategy Approach*, Computer Vision, Graphics and Image Processing, Vol. 32, No. 1, October, 1985, 104-126.
- [Lowe84] Lowe, D.G., *Perceptual Organization and Visual Recognition*, Ph.D Dissertation, Report No. STAN-CS-84-1020, Department of Computer Science, Stanford University, Stanford, California, September, 1984.
- [Matsuyama84] Matsuyama, T., *Knowledge Organization and Control Structure in Image Understanding*, Proceedings of the Seventh International Conference on Pattern Recognition, Montreal, 1984, 1118-1127.
- [Nagao82] Nagao, M., *Control Strategies in Pattern Analysis*, Proceedings of the Sixth International Conference on Pattern Recognition, Munich, 1982, 996-1006.
- [Nazif83] Nazif, A.M., *A Rule-Based Expert System for Image Segmentation*, Ph.D Dissertation, Department of Electrical Engineering, McGill University, Montréal, March, 1983.
- [Roberts65] Roberts, L.G., *Machine Processing of Three-Dimensional Solids*, in Optical and Electro-optical Information Processing, ed. M. Tippet, MIT Press, Cambridge, Massachusetts, 1965, 159-197.
- [Rosenfeld82] Rosenfeld, A., *Image Analysis: Progress, Problems and Prospects*, Proceedings of the Sixth International Conference on Pattern Recognition, Munich, 1982, 7-13.
- [Shapiro83] Shapiro, L.G., *Computer Vision Systems: Past, Present, and Future*, in Pictorial Data Analysis, ed. R.M. Haralick, Springer-Verlag, Berlin, 1983, 199-237.
- [Sowa84] Sowa, J.F., *Conceptual Structures, Information Processing in Mind and Machines*, Addison-Wesley, Reading, Massachusetts, 1984.
- [Waterman, Hayes-Roth&Lenat83] Waterman, D.A., Hayes-Roth, F. and Lenat, D., eds., *Building Expert Systems*, Addison-Wesley, Reading, Mass., 1983.
- [Zucker, Rosenfeld&Davis75] Zucker, S.W., Rosenfeld, A. and Davis, L.S., *General Purpose Models: Expectations About The Unexpected*, Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, 1975, 716-721.
- [Zucker81] Zucker, S.W., *Computer Vision and Human Perception, An Essay On The Discovery Of Constraints*, Proceedings of the Seventh International Joint Conference on Artificial Intelligence, Vancouver, 1981, 1102-1116.

Integration of Remotely Sensed Data and Geographic Information Systems

by

David G. Goodenough

Abstract

Canada is heavily dependent upon the effective utilization of its resources. To better manage the nation's resources, resource managers are increasingly turning to computer-based technologies. Two particularly important technologies for resource information management systems are remote sensing and geographic information systems (GIS). Operational resource managers are using the geographic information systems to store digital representations of their resource maps. Associated with these graphical digital maps are databases containing the attributes of map features.

The major (in terms of contribution to the GNP) land-based renewable resources are forestry and agriculture. In agriculture, the resource cover changes annually and should be monitored frequently during the growing season. For forestry, the changes are slower, but some provinces require annual updates of their forest inventories. Geographic information systems for forestry are loaded by manually digitizing existing forest cover maps. Approximately 6,000 1:20,000 scale maps are required to cover British Columbia, for example. Traditionally, the updating of maps has required re-flying the area of interest. This is a costly procedure. The aerial photos do provide, however, high resolution imagery. Cost savings in geographic information system updating can be achieved by incorporating remote sensing data from satellites.

Since 1978, we have conducted experiments to integrate remote sensing data from satellites and aircraft. We have also investigated the integration of these data with geographic information systems. The experiments have shown that this integration is difficult for several reasons. The remote sensing data may not have sufficient spatial resolution to show the features of interest. The remote sensing data is geometrically corrected in Canada using federal NTS maps, usually 1:50,000 scale. There are substantial (up to 200m) geometric errors between the provincial maps of some provinces and the federal maps. Labelling of some ground features in the GIS may be inconsistent. As a result, we have concluded that artificial intelligence techniques are required to handle the combinatorial explosion of problems reducing the effectiveness of integrating remote sensing data and GIS.

In this presentation, we review the efforts in integrating remote sensing data and GIS and present the approach at the Canada Centre for Remote Sensing. A brief discussion of the problem of exchanging data amongst geographic information systems will also be addressed.

Image segmentation based on color and texture gradient

Phu Thien Nguyen

Scientific Center , IBM France
36 Ave Raymond Poincare, Paris 75116

ABSTRACT

An image segmentation scheme based on a model of human perception of color and texture is proposed. It consists of the following steps:

- Spatial segmentation of each RGB images using an edge detector, contour following and closing algorithms;
- Characterization: each intersection of the segments is now characterized by:
 - color: statistical parameters of the RGB values of pixels within the intersection;
 - shape: perimeter, area, compactness;
 - orientation;
 - topology: neighbor, inclusion.
- Merging of the elementary segments based on the selection of their attributes.
- Segmentation by texture gradient based on the window correlation technique to identify "coherent texture" regions.

RESUME

Nous proposons une méthode de segmentation d'image basée sur un modèle de vision de la couleur et de la texture qui consiste en les étapes suivantes:

- La segmentation spatiale des canaux Rouge, Vert et Blue par détection des contours avec un opérateur de gradient, le suivi et la fermeture des contours;
- La caractérisation de chaque intersection des segments par:
 - couleur: paramètres statistiques des valeurs des pixels à l'intérieur d'une même intersection;
 - forme: périmètre, surface, compacité;
 - orientation;
 - topologie: voisinage, inclusion.
- L'agrégation des segments élémentaires par la sélection de leurs attributs.
- La segmentation de l'image par le gradient de texture en utilisant la corrélation des fenêtres sur l'image pour identifier des régions avec des "textures cohérentes".

INTRODUCTION

The understanding of the human visual system and the photo-interpretation method adopted by human analyst does help in designing algorithms in image analysis. The approach is not so much to simulate or worst to imitate the human visual system but rather to understand the underlying mechanisms in order to deduce a few general criteria which could be implemented to extract pertinent features and to identify objects of different visual properties.

Different objects on images can be identified by either labelling pixels of similar properties or defining their boundaries.

The methods of region growing, split and merge, clustering are typical of the first approach. The second approach consists of methods which are based on the detection of contour. Most existing method of contour detection are based on the gradient in the gray level of a half tone image. We are proposing a method of image segmentation which is based on the integrated use of the three fundamental properties: color, texture and geometrical properties of elementary segments on the image. Two methods of contour detection are developed one using a general Sobel operator to detect gradient in gray level and other using a correlation operator to detect texture gradient. These contours are followed to define elementary segments. The elementary segments are then aggregated using a clustering method. Elementary segments are then merged based on their geometrical and topological properties.

The method is applied to the SPOT simulation data taken over an area in the Southwest of Paris.

COLOR PERCEPTION

The trichromatic model of color vision (Faugeras 1976, Pratt 1978, Caelli 1981) is based on the differential absorption spectra of the three pigments found in the cones of the retina. The energy absorbed by the three types of cone are:

$$\begin{aligned} L(x,y) &= \int I(x,y, \lambda) l(\lambda) d\lambda \\ M(x,y) &= \int I(x,y, \lambda) m(\lambda) d\lambda \\ S(x,y) &= \int I(x,y, \lambda) s(\lambda) d\lambda \end{aligned}$$

where:

I: intensity of energy at a point (x,y)

$$\begin{aligned} l(\lambda) \\ m(\lambda) : \text{spectral absorption curve} \\ s(\lambda) \\ \lambda : \text{wave length (380 to 800 nM)} \end{aligned}$$

L, M and S can be correlated with the primary colors Red, Green and Blue by:

$$\begin{bmatrix} L \\ M \\ S \end{bmatrix} = U \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where U is given by CIE (Comission Internationale de l'Eclairage) for the case of TV set phosphorous by:

$$U = \begin{bmatrix} .3098 & .6321 & .5818 \\ .1208 & .7665 & .1127 \\ .0042 & .1550 & .8408 \end{bmatrix}$$

The response of the cones seems to be non-linear and L, M and S are transformed into L', M' and S' :

$$\begin{aligned} L' (x,y) &= \text{Log (L(x,y))} \\ M' (x,y) &= \text{Log (M(x,y))} \\ S' (x,y) &= \text{Log (S(x,y))} \end{aligned}$$

A number of experiments seem to suggest that in the visual system, these responses are combined and processed as separate achromatic (A) and chromatic information (C1, C2) which can be expressed mathematically as:

$$A = \alpha L' + \beta M' + \gamma S'$$

$$C1 = \mu 1(L' - M')$$

$$C2 = \mu 2(L' - S')$$

The coefficients are given in detail by Faugeras (1976).

A is usually associated with the brightness of the image.

SPATIAL VISION

The visual signals received by retinal receptors are then transmitted to higher level visual cells or group of cells in the Lateral Geniculate Nucleus, and in the visual cortex.

The visual information is processed in a hierarchy depending on the spatial and spectral responses of these visual cells in the receptive fields at different levels. The phenomenon is not yet fully understood, but the following findings (Nevatia 1982, Caelli 1982, Pratt 1976, Tsotsos 1982, Zucker 1984, Cretez 1984) give some insights to how this process is done:

- In the retina, the photo-receptors are grouped in ganglion cells organized into groups of receptive fields. The number of receptors in the ganglion cells is smallest near the fovea and increases nearer to the periphery. This explains the highest visual acuity at the fovea. The spatial arrangement of these receptive fields is suggested to be concentric, hexagonal and organized into a number of layers. The lateral inhibition effect between cells in the receptive field explains the perception of contrast from the information given by the ratio of the response of the central cells to peripheral cells. This contrast sensitivity is a function of spatial frequency of the viewed objects.
- Chromatic response is higher at a lower spatial frequency than achromatic response.
- Chromatic response is relative and the well-known chromatic adaptation effect explains the ability to adapt to local condition of lighting and color balance.
- The receptive fields in the cortex are elongated and hence the response to visual signal at this level is dependent on both spatial frequency and orientation.
- Visual information processing is most precise near the fovea, and the analysis of a scene will require a specific pattern of foveal movement. The analysis of this pattern gives valuable information on how different objects in the scene are being identified.

SEGMENTATION FROM COLOR

An algorithm was developed to detect contours having a strong local gradient (Asfar 1981). The process consists of:

- Detection of edges using a generalized Sobel operator,
- Selection of points having a local maximum gradient,

- Search for the nearest neighbors of each point, taking into account the gradient direction,
- Construction of the contours from the neighbor image.

The detected contours can be followed and closed to isolate different elementary segments. Each segment is then labelled and their attributes computed.

The spatial segmentation is done separately for each of the primary colors Red, Green and Blue. From these segmented images an image of intersection is created where each intersection is given the average R, G and B values of all pixels contained within the intersection. The spatial segmentation of the SPOT data (fig 1) is shown in fig 3. The segmentation can also be done on the L, M and S or A, C1 and C2 components. The R, G and B components are chosen since it was proved that no significant difference is observed when using different color feature sets (Ohta, 1985).

SEGMENTATION BY TEXTURE GRADIENT

The segmentation by texture is treated separately and a texture gradient operator is proposed based on the window correlation technique. The analysis is restricted to chromatic component A (fig 5) because the eye is more sensitive to spatial resolution.

At the boundary of two areas of different textures, the correlation between two windows is maximum along the boundary and minimum across. By calculating the correlation of a central window to eight neighboring windows, the direction and the amplitude (difference between the maximum and minimum correlation coefficient) of the "texture gradient" can be derived. From these two parameters contours can be traced and followed to isolate "coherent texture" regions.

CLUSTERING

An unsupervised classification based on the L1 distance (Ramirez 1982):

$$d_j = \sum_{i=1}^{N_{band}} |\mu_j(i) - x(i)|$$

where : $\vec{\mu_j}$ is the vector of the center of cluster j
j = 1, 2, ... N_c is the number of clusters.

Pixels are assigned to the nearest cluster and its center updated according to:

$$\vec{\mu_k} = \frac{n_k \vec{\mu_k} + \vec{x}}{n_k + 1}$$

where n_k is the population of cluster k

The clusters found can be:

- split if their standard deviation is below a certain threshold.
- merged if their distance:

$$d_i = \left\{ \sum_{k=1}^N \frac{[\mu_i(k) - \mu_j(k)]^2}{\sigma_i(k) \cdot \sigma_j(k)} \right\}^{1/2}$$

is below a certain value.

A clustering result of the original SPOT data is shown in fig 2 as compared with the result obtained with the segmented images (fig 4).

GEOMETRICAL FEATURE EXTRACTION

The segmented image from either the segmentation by color or texture gradient can be stored under Run Length Coding format (Loodts 1985). The following attributes can then be computed:

Perimeter: The perimeter is defined as the number of pixels belonging to the border of each surface. This is done by accumulating the number of pixels of the same surface surrounded by pixels belonging to other surfaces in two consecutive lines.

Area: this is done by accumulating the length values of pixels belonging to the same surface in two consecutive lines

Compactness: this is a measure given by the parameter $\frac{A}{P^2}$ where P is the perimeter and

A the area of each segment.

Topological features: the relation between segments such as neighborhood, inclusion etc. can be determined by considering pixels belonging to their common boundaries.

ELEMENTARY SEGMENT MERGING

Each elementary segments after the segmentation, clustering and feature extraction are now characterized by a set of parameters:

- color: statistical parameters of the RGB values of pixels within the intersection;
- shape: perimeter, area, compactness, linearity;
- orientation;

- topology: neighbor, inclusion,

The merging can be done by applying a decision based on the selection of these features.

An example of elementary segment merging on the texture classified image (fig 6) by the following criterion:

"Small segments (surface < threshold) included in a large segment (surface > threshold) are merged with the large segment if their forms are not compact ($-\frac{A}{P^2}$ is small)".

The result of this merging is shown in fig 7.

CONCLUSIONS

The image segmentation as proposed integrates the spectral and spatial attributes of each pixels in the identification of each elementary segments on a color image. Furthermore these elementary segments can be aggregated by an intelligent selection of their attributes. This presents an approach nearer to the method of photo-interpretation, but there still exists a number of drawbacks which can only be achieved by further researches in the following points:

- The segmentation process is based on global features and local environment can not be taken into account to cater for:
 - local chromatic adaptation;
 - size invariance : a fixed window cannot be used over all regions of wide spectral frequency variation;
 - orientation invariance:
- definition of texture gradient by correlation is still a very crude approximation;
- the segmentation of an image into regions of different color texture with consistent pattern of primitives having a similar color still remains to be investigated. A combined use of texture and color information by averaging the R, G and B segmented images by using the texture class map as a mask is presented in fig 8, this image shows the colors of regions of "similar degree of homogeneity".
- the hierarchical approach which can locally adapted to the complexity of different regions on the image.

Acknowledgement

The author is most thankful to Mr A. Ballut of IAURIF, Paris for providing the SPOT simulation data over the Paris region.

References

1. Asfar L, "A method for contour detection, segmentation, and classification of Landsat images", International geoscience and remote sensing symposium, Washington D.C., June 1981, pp. 298-304.
2. Cretez J P, L Tanimoto, "Perceptual constancy and the multi-layer visual model: position and size invariances", 7th Int conf on Pattern recognition, Montreal, July 30-Aug 2, 1984,, pp. 518-520.
3. Caelli T, "Visual perception" Pergamon Press, Oxford, 1981, pp. 172-183.
4. Faugeras O, "Digital color image processing and psychophysics within the framework of a human visual model", Ph D dissertation, Unive. Utah, Salt Lake City, June 1976.
5. Loodts J and P.T. Nguyen, "The use of Run Length Coding technique in image storage and processing- an application in remote sensing", The 4th Scandinavian Conference on Image Analysis, Trodheim, June, 1985
6. Nevatia R., "Machine perception", Prentice Hall, New Jersey;1982, pp.90-99.
7. Ohta Y., "Knowledge-based interpretation of outdoor natural scenes", Pitman, Boston, 1985.
8. Pratt W K, "Digital image processing", John Wiley and Sons, New York, 1978, pp. 43-47.
9. Ramirez F, "Local Operation at Hacienda" IBM-UAM Scientific centre, Madrid, Feb 1982.
10. Tsotsos J K, "Knowledge of the visual process: content, form and use", 6th Int Conf on pattern recognition, Munich 1982, pp. 654-668.
11. Zucker S W, A Humel, "Receptive fields and the representation of visual information", 7th Int conf on Pattern recognition, Montreal, July 30- Aug 2, 1984,, pp. 515-517.

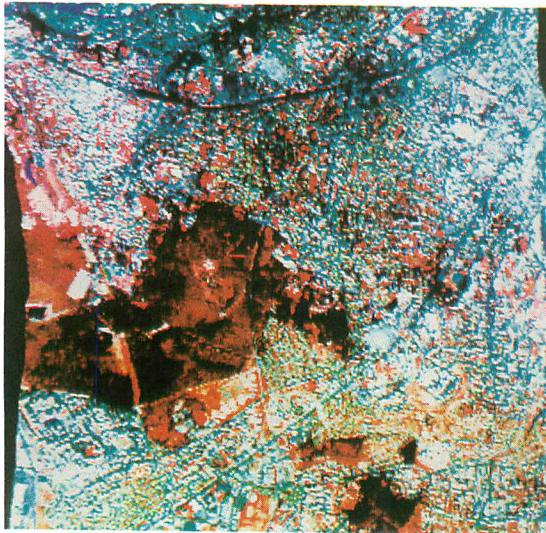


Fig. 1. Color image of the Red, Green and Blue components.

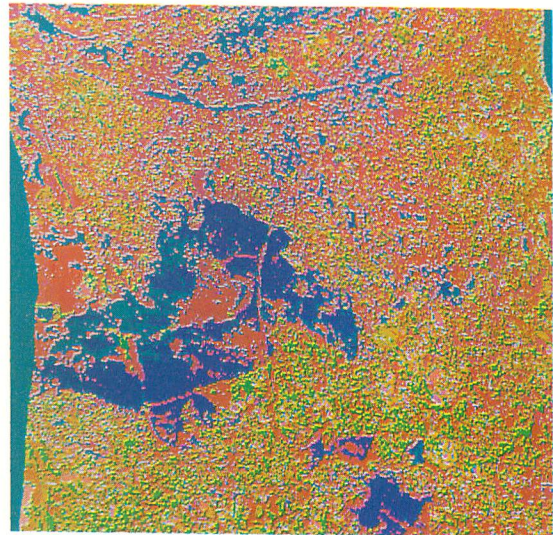


Fig. 2. Classified image of fig 1 by clustering.

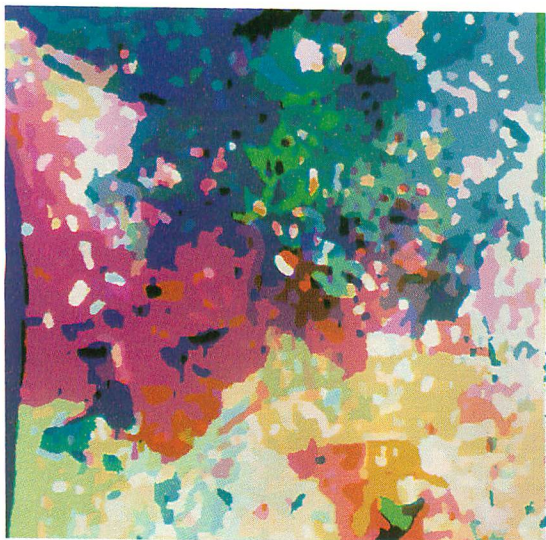


Fig. 3. Color image of the intersection of the three segmented images.



Fig. 4. Classified image of fig 3 by clustering.

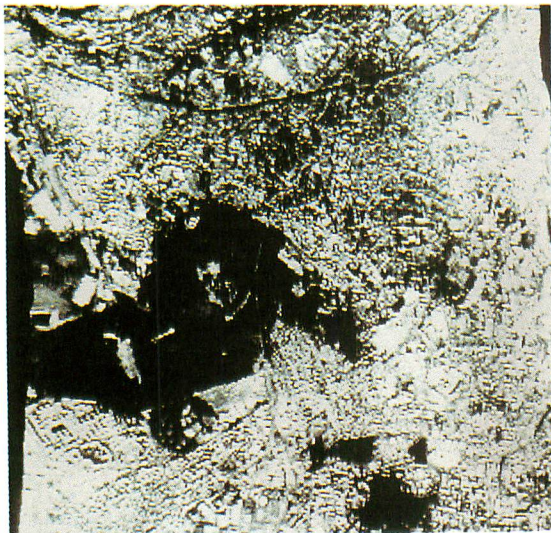


Fig. 5. Achromatic component
of the Red, Green and Blue images.

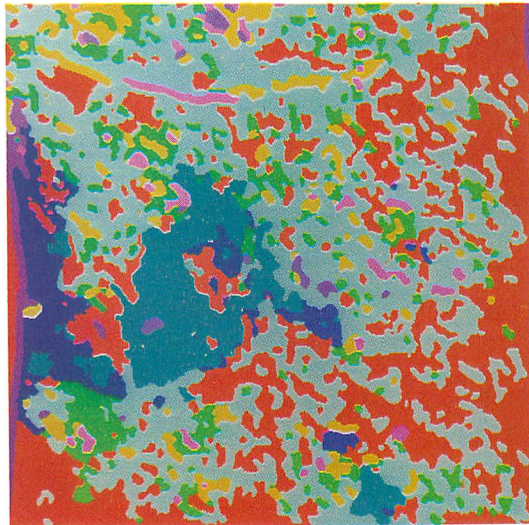


Fig. 6. Classified image of the segmented image
by texture gradient after clustering.

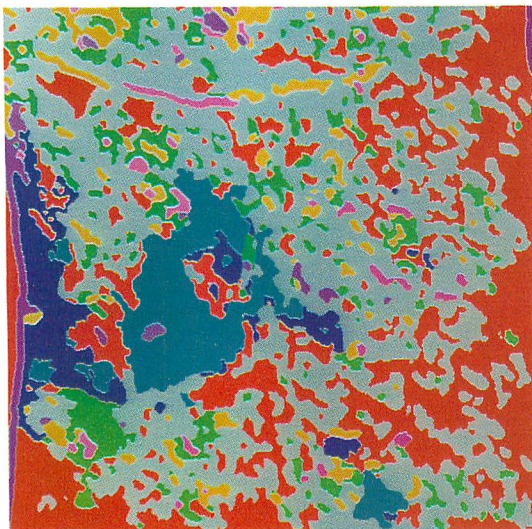


Fig. 7. The result of elementary
segment merging of fig 6.

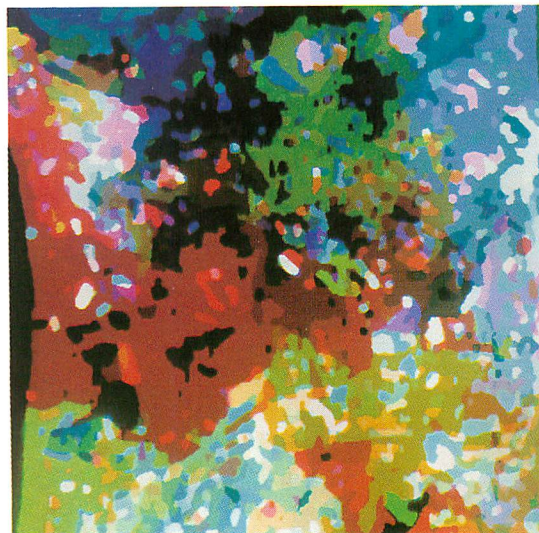


Fig. 8. The result of selective smoothing
of fig 3 using fig 7 as a mask.

MAP/IMAGE CONGRUENCY EVALUATION KNOWLEDGE BASED SYSTEM

Gordon W. Plunkett and David G. Goodenough
Canada Centre for Remote Sensing
Ottawa, Ontario, Canada

Morris Goldberg
University of Ottawa
Ottawa, Ontario, Canada

ABSTRACT

A Knowledge Based System (KBS) for analyzing LANDSAT MSS images and comparing this analysis to corresponding geocartographic data is presented. This paper discusses the preprocessing requirements for the LANDSAT and the geocartographic data for a uniform representation of the data. The segmentation of the LANDSAT data and the interpretation of the segments are presented. The preprocessed data are read into the Map/Image Congruency Evaluation (MICE) KBS where the image segments are classified and then compared with the map data, based on class, segment size, shape, and location. Results of the map/image congruency analysis are output and converted to image form. This paper presents the MICE KBS and reviews the results generated for a LANDSAT MSS scene of the Prince George area of British Columbia.

KEYWORDS: LANDSAT, computer cartography, image analysis, artificial intelligence, knowledge based systems

SUMMARY

Remotely sensed data, particularly from the LANDSAT series of satellites, are being used for a wide variety of useful applications. One of the more challenging applications is the data integration of remote sensing data with existing cartographic data bases. It was found that simple algorithmic data integration methods did not provide satisfactory results due to various geometric irregularities in the remote sensing data and in the cartographic data. These spatial irregularities could be due to factors such as temporal differences between the data, spatial errors in the map data or topographic effects in the remote sensing data. Algorithmic techniques break down in this data integration [BILLINGSLEY82]. Therefore, we have tried to solve the integration problem with a knowledge based system approach.

The Map/Image Congruency Evaluation (MICE) knowledge based system was developed to study the spatial differences between maps and images. Map/image congruency evaluation means the determination of the spatial agreement of features in the map with the corresponding feature in the image. The data integration problem study included three basic operations. These operations were: (i) preprocessing the data for uniform representation of both

the image and the map data; (ii) spatial reasoning on the data using the PROLOG-based MICE system; (iii) output of a congruency evaluation map from the results of the MICE analysis.

The MICE system was evaluated using LANDSAT MSS data for the Prince George area of British Columbia and a BC provincial forest cover map. Various levels from the BC digital map were selected. These levels corresponded to single-line creeks and rivers; double-line rivers and lakes; road and utility systems; and the forest cover. Each level was gridded to a 50x50 metre grid. The LANDSAT data were geocoded by the Canada Centre for Remote Sensing (CCRS) Digital Image Correction System (DICS) to a UTM coordinate grid with 50x50 metre pixels. The sub-area of the image corresponding to the map was selected.

The LANDSAT image was then segmented to highlight the various features. Numerous properties such as the segment shape, size, location and spectral means were evaluated. The map data were similarly processed to determine properties such as shape, size and location. These data were then read into the knowledge based MICE system.

The MICE system then evaluated the matching of the various segments from the map and image by examining the identification of the segments and the structure of the segments. The identification of the segments was done to determine if the structurally corresponding segments have corresponding identifications. For instance, a segment that has been identified as a lake in the map data must correspond to a segment in the image that has a spectral signature that corresponds to a lake. If the LANDSAT segment does not have a corresponding spectral signature, then the segment is only weakly identified. Finally, the exact positions of the remaining segments are determined and all location differences are reported.

The MICE system, which is currently under development, uses a variety of meta-level rules and object-level rules. These rules and some of the internal workings of the knowledge based system will be given, as well as suggested enhancements.

INTRODUCTION

For many years, human photo-interpreters have been analyzing air-photos, deciding on the classi-

fication of various objects in the photo and then transcribing the classification and location of these objects onto a map or more recently into a geographic information system [ZARZYKI82]. Since this map making procedure is primarily a human endeavour, it is prone to human error. In addition, the world land-mass is a constantly changing entity. For example, rivers meander further, forests burn or are cut, and subdivisions and roads are built. The cartographic data on the other hand, is relatively static and is only updated periodically.

For some time, the remote sensing community has been extolling the virtues of the integration of remote sensing data with Geographic Information Systems (GIS). This data integration problem has been researched and solutions developed, which are used operationally by some agencies [HEGYI83]. However, the automatic integration of remote sensing data with geographic information systems is not yet possible as it still requires human interpretation and assistance.

One of the first steps in the integration of remote sensing data with GIS data is simply to evaluate how similar or different are the map and image data. It has been shown that algorithmic techniques such as differencing and correlations simply don't work very well [PARSONS84] [GOODENOUGH85]. Also many rule based systems for image interpretation have shown promising results [MCKEOWN85]. Thus, a knowledge based system for the comparison or congruency evaluation of maps and images was developed.

The MICE system was developed on a VAX 11/780 system running VMS. The VAX system hosts a suite of software from Intergraph Corp. for processing and manipulating cartographic data. Also, the VAX hosts a large suite of image processing software, that was developed in VMS Fortran at CCRS. It was decided that the existing software base be used for some of the processing programs. Additional processing and reformatting programs were developed in Fortran and the results fed into the MICE KBS. MICE itself, was written in M-PROLOG from Logicware Inc.

The Prince George area of British Columbia was selected as a test area as a number of data sets from a variety of sources were available. The digital cartographic data from the BC Ministry of Forests was obtained. These data contained the hydrography, cadastral, forest, roads, railroads and other cartographic information required for a forest cover map. The map scale was 1:20,000 and corresponded to the UTM map number 93G096.

The LANDSAT MSS geocoded image for the Prince George area (93G15) was obtained from CCRS. This DICS product [GUERTIN81] consists of the LANDSAT data scaled and projected onto a 50 metre grid, in a UTM projection.

GIS PREPROCESSING

The BC forest cover map was received and stored as an Intergraph design file. This file contained a variety of cartographic information, but for the purposes of this experiment, only the following information was processed:

Information

File Level

a) single-line creeks, rivers	5
b) double-line rivers, lakes	6
c) utility systems	8
d) forest cover typelines	9

The levels were extracted and any spurious information or text was deleted. Each level was edited using an automated technique for ensuring that all line intersections were cartographically sound. Next, the levels were individually converted from vector format to grid format, based on a presence/absence algorithm onto a 50x50 metre grid. These grid files were then converted to CCRS standard imagery files. Each polygon (such as a lake), which was not fully filled was filled. The image was precision registered to a UTM grid and each entity of the map was identified uniquely and its location was run length encoded. Finally, each unique element, along with its run length encoded location was converted into symbolic object form. The file containing these symbolic objects were read into the MICE system. The procedure for preprocessing the cartographic data is given in Figure 1.

IMAGE PREPROCESSING

The LANDSAT MSS image (frame Id: 50458-18360) used for this experiment was imaged on June 25, 1985. The MSS image was then precision geocoded on the CCRS DICS system. The area of the image corresponding to the BC forest cover map 93G096 was extracted and precision registered to the rasterized 93G096 map data. The MICE system then requires the segmentation and statistical analysis parameters of these segments.

The image subscene is first operated on by a specified gradient operation. The resulting file is then segmented [BOILEAU85]. Each segment is uniquely identified and its location run length encoded. Next statistical information on each segment is generated. This statistical information is shown in Table 1. The segment locations and segment statistical values are converted into symbolic object format for input to the MICE KBS. The procedure for preprocessing the image data is given in Figure 2.

MAP/IMAGE CONGRUENCY EVALUATION KNOWLEDGE BASED SYSTEM

The map/image congruency evaluation knowledge based system is implemented in PROLOG using a shell for developing hierarchical expert systems for remote sensing [GOLDBERG85] [BRAUN85]. The implementation is primarily divided into two rule types. These are the meta-rules, which are rules about what MICE should do next, based on information deduced to that point. The other type of rule is the object rule, which is a rule that has been input to the MICE system, or has been deduced by the MICE system.

Meta-Level Rules:

The meta-rule consists of four items. These items are: 1) condition predicate; 2) action procedure; 3) phase number; 4) rule number.

FIGURE 1:
THE PROCEDURE FOR PREPROCESSING THE
CARTOGRAPHIC INFORMATION

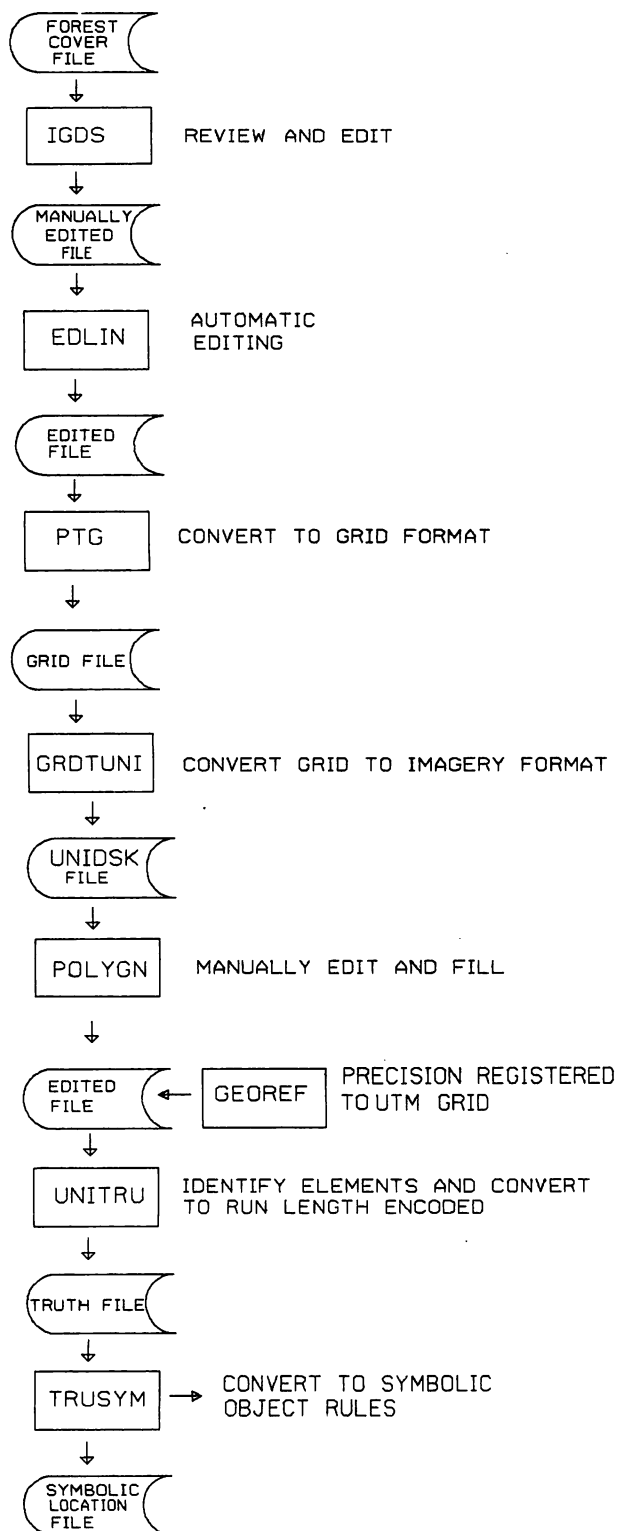
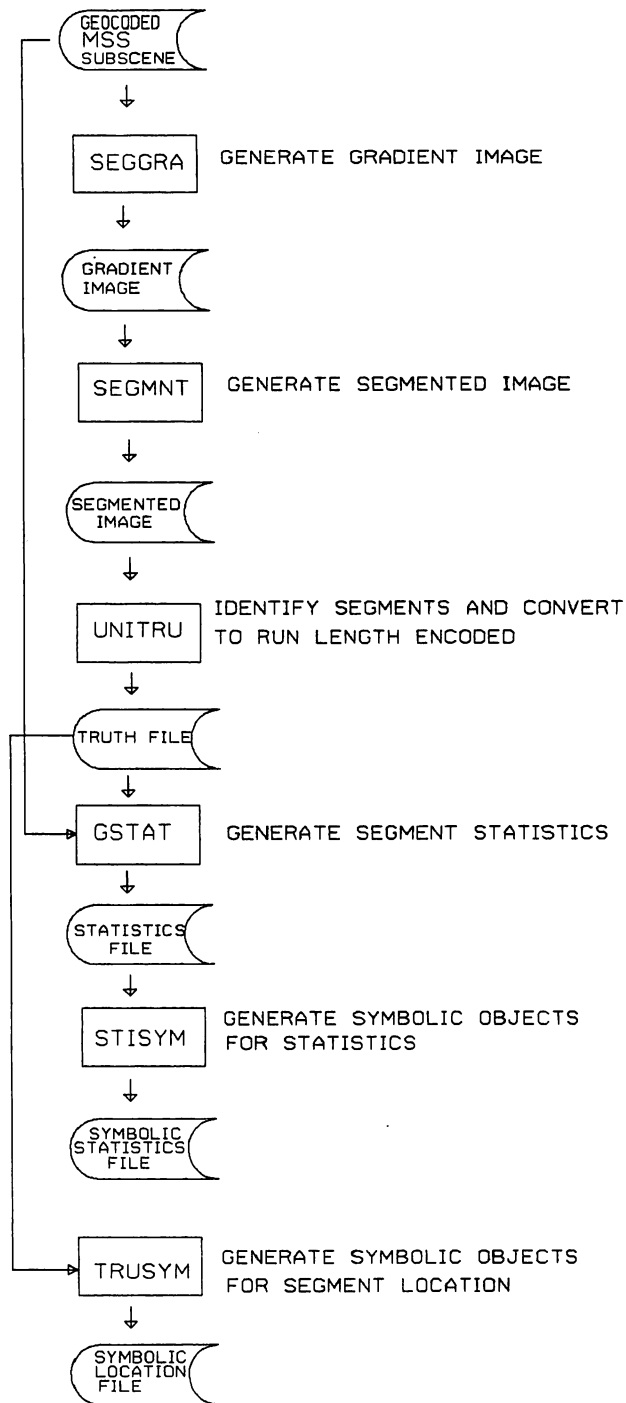


FIGURE 2:
THE PROCEDURE FOR PREPROCESSING
LANDSAT MSS IMAGE



The condition predicate (or "if" part of the rule) is evaluated by MICE to determine if the condition predicate is true. The action procedure (or "then" part of the rule) may then be executed if the condition predicate is true. The phase number is the strategy level within the meta-level procedure in which this rule is to be evaluated. The rule number is simply to uniquely identify each rule in the meta-level procedure. An example of two meta-level rules for one phase from MICE is as follows:

```
if:    the image segments were identified (ok)
then:  compare map and image segment sizes
phase: 9
rule #: 17
if:    not (the image segments were identified
         (ok))
then:  write the string ("no image segments
         identified")
phase: 9
rule #: 18
```

Object Rules:

Object rules also consist of four items. These are: 1) condition predicate; 2) action procedure; 3) rule number; 4) certainty factor. These rules deduce object values based on the values of the objects in the condition predicate. The rule number uniquely identifies the rule number and the certainty factor is a value from 0 to 100.

Objects:

Objects are the basic manipulation element of MICE upon which deductions are made. Objects consist of four values, which are: 1) object context or description; 2) object attribute; 3) object value; 4) measures of belief and disbelief. MICE uses context values such as: source (image-MSS), source (map-bcfs), segment (segment-number) and class (class-name). Attribute values such as location, size, mean-channel, and shape etc. are used with the corresponding value of the attribute in the object element. The measures of belief and disbelief for each element are included. The measures of belief and disbelief range from 0 to 100. A belief/disbelief value of 100 means that this object is very believable/disbelievable. A smaller value indicates less belief/disbelief in this object. A sample object element is as follows (in PROLOG notation):

```
obj([[*, source (image-MSS), *, segment (2), *,
class (hydrography), * ], size, [2160], [75,25]]).
```

This Prolog statement means:

- 1) The source of the segment is the LANDSAT MSS image.
- 2) The segment is segment number 2.
- 3) The segment has been classified into the class hydrography.
- 4) The attribute of this object element is the size of the segment.
- 5) The attribute value or the size of the segment is 2160 pixels.
- 6) The measure of belief for this object is 75 and the measure of disbelief is 25.

A simplification of the agenda that MICE uses to perform the congruency evaluation is as follows:

- 1) load the map object elements
- 2) load the image object elements
- 3) perform preliminary classification on image objects
- 4) get the next map segment
- 5) find all image segments near map segment (focus)
- 6) compare class values
- 7) compare segment sizes
- 8) compare segment shapes
- 9) compare segment locations
- 10) output results

The output generated by MICE is in the form of object elements, that were deduced by the KBS. These object elements indicate where the map and image segments overlap, where they are partitioned, where they are hierarchical and where they are bipolar. These elements are then converted from symbolic form to run length encoded format. Finally the results are converted to imagery format, which can be displayed and reviewed.

SAMPLE OUTPUT

An experiment using the MICE KBS was performed using LANDSAT MSS and digital map data from the BC Ministry of Forests. The sample outputs are for the double-line rivers and lakes data from the BC map. The figures 3 to 10 show the input and the output from various phases within the MICE system. They all correspond to the processing of the map or image data from figure 3 or figure 5, respectively.

Figure 3 shows the input LANDSAT MSS image for band 7 (infrared 0.8µm to 1.1µm) for the Prince George area of BC. The image has been geocoded to the UTM projection and resampled to 50m x 50m pixels. The date of the image is June 2, 1985. Figure 4 shows the same image following the application of the Sobel gradient operator, segmentation, and grey level coding. The coding algorithm is for display purposes only. It reviews the segments and assigns each segment a digital value that ensures that no neighbouring segment has the same value (grey-level). However, non-neighbouring segments may have the same grey-level value. Figure 5 shows the input map vector data for the double-line hydrology level of a BC forest cover base map. Figure 6 shows the same map data after it was cleaned and rasterized. Cleaning means removing annotation, processing vector and points (overshoot/undershoot conditions), processing vectors against themselves (knot conditions), and processing vectors against other vectors (lobe conditions). Rasterization uses the presence/absence algorithm.

Figure 7 shows the segments of the image that were identified as being in a map segment window (focused) and were also classified as hydrology. A focused image segment means that the image segment is "near" the map segment. The map segment window is the smallest rectangle that encloses the segment. An image segment is near (focused on) the map segment if any part of it is within the window or half the window length in any direction.

Figure 8 shows the focused image segments that are of similar size. Figure 9 shows the focused image segments with similar shape and Figure 10 shows the focused image segments which are

determined to be overlapping segments. An image segment that is of similar size to a map segment satisfies the following rule

$$50 < \frac{\text{map segment size}}{\text{image segment size}} \times 100 < 150$$

An image segment that is of similar shape to a map segment satisfies the following rule

$$50 < \frac{\text{map segment shape}}{\text{image segment shape}} \times 100 < 150$$

An overlapping image segment is one where any pixel of the image segment overlaps any pixel of the map segment.

CONCLUSIONS

The results reported thus far are very encouraging for the use of knowledge based systems for performing visual tasks, such as verifying the congruency of maps and images. Obviously, this is the first step in automating the process of integrating remote sensing data with geographic information systems. Further work is required and more rules must be added to enhance the functional performance of the congruency verification procedure, but the same techniques should also apply then to the extraction of selected areas in the image and including this information in the GIS system. Future work will include experiments with LANDSAT Thematic Mapper data and federal topographic maps.

REFERENCES

- [BILLINGSLEY82] Billingsley, F.C., "Edited oral presentation", in Proceedings of the NASA Workshop on Registration and Rectification, Bryant, N.A. (ed), NASA-JPL, 1982.
- [BOILEAU85] Boileau, J.M., Goldberg, M., "A Selection of Segmentation Similarity Measures for Hierarchical Picture Segmentation", Graphics Interface, p.179-186, Montreal, May, 1985.
- [BRAUN85] Braun, F., University of Ottawa, MASC Thesis, 1985.
- [GOLDBERG85] Goldberg, M., Goodenough, D.G., Alvo, M., and Karam, G.M., "A Hierarchical Expert System for Updating Forestry Maps with LANDSAT Data", Proceedings of the IEEE, Vol.73, No.6, June, 1985.
- [GOODENOUGH85] Goodenough, D.G., Fung, K.B., Hegyi, F., Robson, M., and Swanberg, N.A., "Integration of Geographic Information Systems with LANDSAT Thematic Mapper Data", paper presented at IGARSS'85.
- [GUERTIN81] Guertin, F.E., Shaw, E., "Definition and Potential of Geocoded Satellite Imagery Products", Presented at the 7th Canadian Symposium on Remote Sensing, Winnipeg, Manitoba, Sept. 1981.
- [HEGYI83] Hegyi, F., Quenet, R.V., "Integration of Remote Sensing and Computer Assisted Mapping Technology in Forestry", Paper presented to The Canadian Institute of Surveying Congress, Victoria, B.C., 1983.
- [MCKEOWN85] McKeown, D.M., Harvey, W.A., McDermott, J., "Rule-Based Interpretation of Aerial Imagery", PAMI-7, No.5, Sept. 1985.

- [PARSONS85] Parsons, T.J., "Towards Robust Image Matching Algorithms", SPIE, Vol.504, Applications of Digital Image Processing, VII (1984).
- [ZARZYCKI82] Zarzycki, J.M., and Allam, M.M., "Canadian Council on Survey and Mapping - National Standards for the Exchange of Digital Topographic Data", Topographical Surveys Division, Surveys and Mapping Branch, April, 1982.

TABLE 1: Image Statistical Information Generated for MICE Input

1)	segment size (number of pixels)
2)	segment MSS channel 1 (band 4) mean
3)	segment MSS channel 2 (band 5) mean
4)	segment MSS channel 3 (band 6) mean
5)	segment MSS channel 4 (band 7) mean
6)	segment MSS channel 1 (band 4) maximum value
7)	segment MSS channel 2 (band 5) maximum value
8)	segment MSS channel 3 (band 6) maximum value
9)	segment MSS channel 4 (band 7) maximum value
10)	segment MSS channel 1 (band 4) minimum value
11)	segment MSS channel 2 (band 5) minimum value
12)	segment MSS channel 3 (band 6) minimum value
13)	segment MSS channel 4 (band 7) minimum value
14)	segment shape (perimeter ² /area)
15)	segment window (smallest rectangle containing segment)

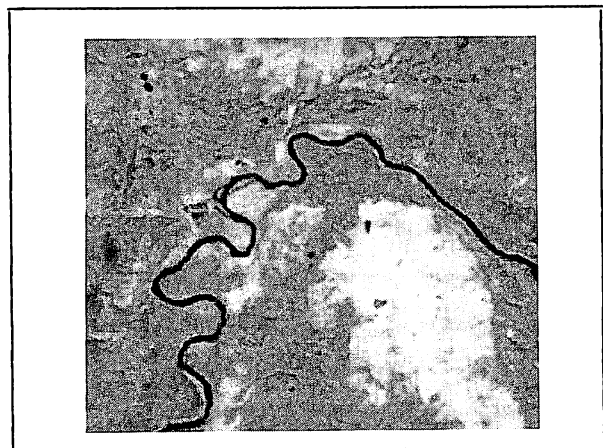


FIGURE 3 LANDSAT MSS band 7 image for Prince George area BC

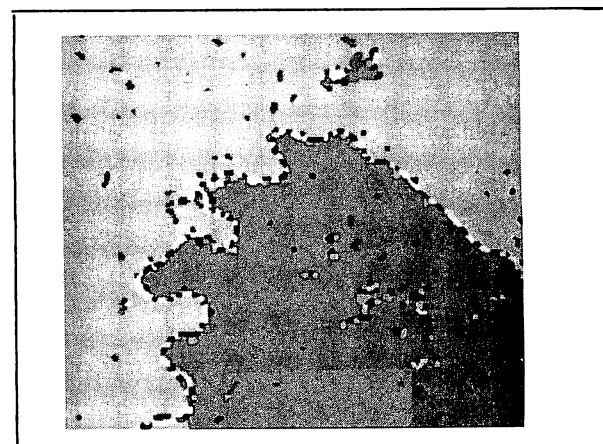


FIGURE 4 LANDSAT image following Sobel gradient operator, segmentation and coding for display. This image corresponds to the LANDSAT image of Figure 3.

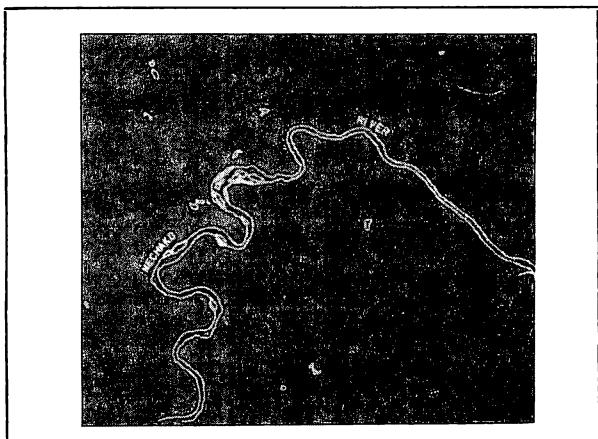


FIGURE 5 BC Ministry of Forests base map for doubleline hydrography vector data.

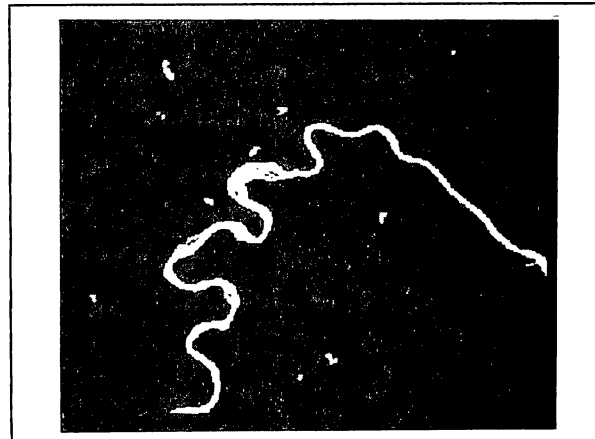


FIGURE 6 Map data following cleaning and rasterization of vector data from Figure 5.

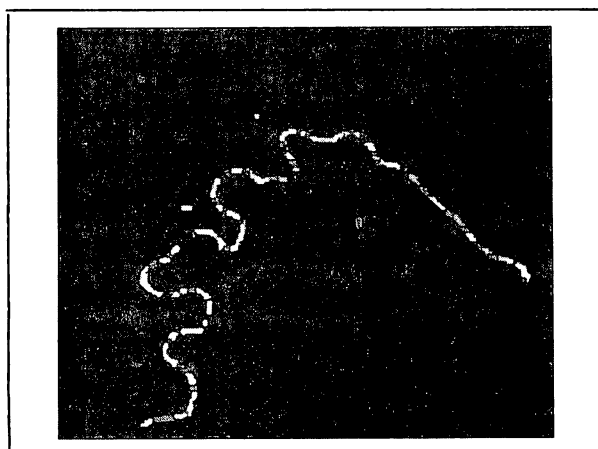


FIGURE 7 Image segments (from Figure 4) that were classified as hydrography and are near any map segment.

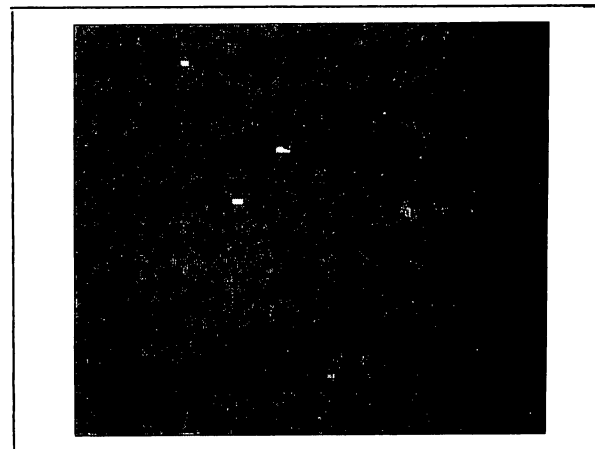


FIGURE 8 Image segments (from Figure 4) that were of similar size to any map segment that was near.

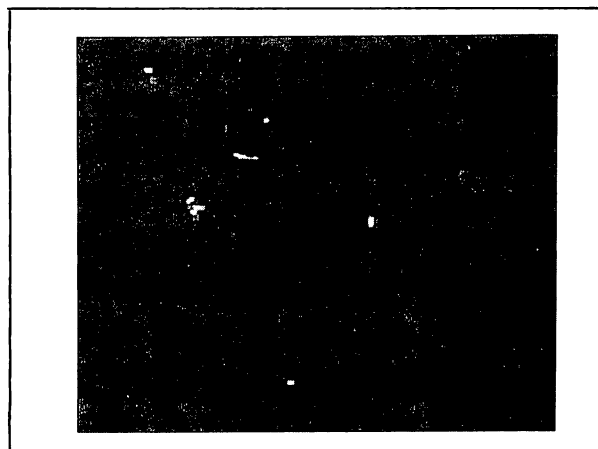


FIGURE 9 Image segments (from Figure 4) that were of similar shape to any map segment that was near.



FIGURE 10 Image segments (from Figure 4) that were overlapping any map segment.

A CONTEXT BASED TECHNIQUE FOR SMOOTHING OF DIGITAL THEMATIC MAPS

Ben Yee
Dean Turpin
MacDonald, Dettwiler and Associates Ltd.
Richmond, B.C.

Evert Kenk
Mark Sondheim
B.C. Ministry Of Environment
Victoria, B.C.

ABSTRACT

This paper describes a context based technique for smoothing digital thematic maps produced by multispectral classification of Landsat Thematic Mapper data. The output of this technique is a "maplike" product which can be directly used as input to a geographic information system.

Keywords - classification, remote sensing, geographic information systems, image analysis.

INTRODUCTION

The work described here is the result of ongoing research at MacDonald, Dettwiler and Associates Ltd. into techniques for automated mapping utilizing remotely sensed data.

Multispectral classification techniques have been used on Landsat data to produce landcover maps. Traditional pixel-by-pixel multispectral classification techniques generally result in noisy or "speckly" images with a large number of small polygons which complicate the image and make the thematic image difficult to interpret. When classified imagery is used as an input to geographic information systems, the complexity of the classified image does not facilitate ease of map update or production. Thus there is a need for an effective technique to convert the classified image to a more cartographically acceptable product.

The role of a map is to effectively present information to users for their specific application at a given level of detail. In traditional mapping, a minimum mapping unit criteria is frequently used to simplify the map. In developing the final map, a cartographer takes into consideration contextual and esthetic factors.

Context plays an important role in cartography. It has been found that a large degree of the "errors" in automated computer classification is one of contextual interpretation. Even if an image could be classified with 100% accuracy, if it does not correspond to the interpretation desired by the mapper it would have "error". Interpretation frequently depends on the size, shape, and context. A small clearing

inside a forest may still be interpreted as "forest" whereas a similarly sized bog inside a forest may be interpreted as "bog".

In this paper we demonstrate a technique which incorporates context in smoothing the digital thematic map to produce a more "maplike" product.

REVIEW OF EXISTING TECHNIQUES FOR SMOOTHING DIGITAL THEMATIC MAPS

Several different techniques for smoothing classified images have been documented including majority and minimum area filtering techniques [DAV76, SCH83].

In the majority filtering approach, the center pixel of an N-by-N neighborhood is replaced by the majority class of the neighborhood so small isolated polygons will be eliminated. Although this technique significantly reduces the number of polygons, small polygons still remain as there is no explicit control over the minimum polygon size. With larger window sizes there is a tendency for the majority filter to erode smaller features and affect the integrity of the polygon boundary location. Smaller window sizes however, do not produce adequate smoothing.

In the minimum area filtering approach, the class of an undersized polygon is converted to the the class of the polygon with which it shares the largest common boundary. Since there is no explicit control of the class conversion of undersize polygons, there may be undesired class conversions when an undersized polygon is converted to a very dissimilar class rather than to a more similar neighbor.

INCORPORATING CONTEXT IN SMOOTHING OF DIGITAL THEMATIC MAPS

Recognizing the importance of context in mapping we conclude that an effective technique for smoothing thematic maps should attempt to incorporate context.

As an example of the importance of context, we cite criteria used by the B.C. Forest Service in their production of forest cover maps [FOR82]:

1. Minimum Area Criteria:

- "Minimum area may be fixed or variable, with clearly defined, important stands being mapped down to smaller areas than those which are less well defined and less important."
- "Minimum type size of approximately 1.5 cm and 1.0 cm for forest and nonforest land respectively are recommended regardless of photographic scale."
- Exceptions are made to the minimum area criteria in certain contexts (see 3).

2. Avoidance of Complicated Shapes:

- "Connect small types with similar structure whenever possible."
- "Avoid complicated, irregular type lines that hinder plotting, reading of the map, and history updating."

3. Context:

- "Type out small nonforest patches isolated within high value types and, conversely, high value patches of timber within low value types."

We base our technique for smoothing of digital thematic maps on the concept of minimal mapping areas and use context to guide conversion of undersize regions. Regions are defined as a contiguous group of pixels of the same type. A region is undersized if it occupies an area less than that specified for its class. By allowing individual minimum sizes for each class and user specifiable preferences for class conversion we can incorporate spatial context, the significance of the land cover type, and the context of the end application into the smoothing process.

Specifically the criteria that we use in our contextual smoothing technique are:

1. Regions must be larger than the specified minimum mapping size for its class or type (see Table 1).
2. If an undersized region is surrounded then it is merged with the surrounding region.
3. Regions below the minimum size that are not entirely surrounded by a single class are merged with a neighbour whose class is most similar to the class of the undersize region. Table 2 shows a similarity matrix in which higher numbers represent an increasing degree of similarity.

4. If there is equal preference of a merge, merge with the region which shares the larger common boundary.

The minimum size of the regions and class similarities are specified by the user for the type of land cover being mapped, the mapping scale, and the end application of the map product.

Detailed explanation of the implementation of the technique is beyond the scope of this paper. In our work, we utilize a vector representation of the regions as the basis for our operations.

TEST RESULTS

Evaluation of the contextual smoothing technique was performed on thematic maps of two study areas in British Columbia, Canada: Adam River on Vancouver Island and Cranbrook in South Eastern British Columbia. The land cover maps produced from the classification of the Landsat scenes are for subsequent use in wildlife habitat mapping and thus the interpretation or smoothing of the land cover map was tailored for this purpose.

The Adam River and Cranbrook study areas were classified into 12 and 16 land cover classes respectively, using supervised maximum likelihood classification of Landsat 5 Thematic Mapper data. The results are shown in Figures 1 and 5.

Prior to contextual filtering, a 3-by-3 majority filter is applied to eliminate small isolated groups of pixels (see Figure 2). Minimum polygon sizes and similarity matrices suitable for the application are then specified. Tables 1 and 2 show the minimum polygon sizes and similarity matrix used in the Adam River study area. Figures 4 and 6 show the results of contextual filtering on the classified images shown in Figures 1 and 5.

Comparing the contextually filtered images to the raw classified images, we see a significant simplification of the images with a great reduction in the number of polygons. Table 3 shows a relative comparison of the number of polygons after the different smoothing operations for the Adam River study area. Comparing the results of the 5-by-5 majority filter (see Figure 3) to those of the contextual filter, we see that although the larger majority filter smooths the image significantly, it still leaves small insignificant polygons which the contextual filter has eliminated.

An important criteria in evaluating the smoothing technique is the accuracy of the resulting output. Evaluation of the accuracy of contextual smoothing (see Table 4) shows that the technique results in accuracies similar to those of large majority filters. The boundary locations of the polygons which result from this technique tend to be more accurate than those from majority filtering due to a tendency for the majority filter to erode smaller

features. Furthermore, with a contextual basis, the conversion of the undersize classes will in general be logically more accurate than with techniques which do not take context into account.

Most importantly, the result is a product which is more "maplike", understandable, and compatible with geographic information systems.

Although not illustrated in this paper, an additional contextual criteria, which considers the significance of the polygon is considered, was tested and could be useful in some applications. The basic concept was that smaller polygons should be retained when it is of high significance than when it is of low significance. More simply stated, the minimum region sizes can change depending on the context; a smaller minimum size is warranted where the most similar adjacent polygon is significantly different (in our case we used a similarity score of less than 3) than in the case where the adjacent polygons are very similar. This criteria was utilized in further generalization of the image and it was found with that significant features were still retained.

CONCLUSION

In this paper we have shown an effective technique for converting classified image into a more cartographically acceptable product. The result is an output that can be vectorized and directly input to a geographic information system without further digitization or generalization.

We have shown the importance of context in mapping and demonstrated how context can be incorporated in a technique for smoothing of multispectral classification.

This technique is particularly effective for filtering of multispectral classified imagery, but can be equally effective when applied to other types of digital thematic maps.

ACKNOWLEDGEMENT

This work was performed under the financial support of an Industrial Research Assistance Program grant "Remote Sensing #2009".

REFERENCES

- [DAV76] Davis, W.A. AND F. Peet., The Identification and Reclassification of Small Region On Digital Thematic Maps, Forest Management Institute Information Report FMR-X-90, 1976.
- [FOR82] Forestry and Range Inventory Manual, B.C. Ministry of Forests Inventory Branch, 1982.
- [SCH83] Schowengerdt, R.A., Techniques for Image Processing and Classification in Remote Sensing, Academic Press, New York, 1983.

[STR83] Strahler, A.H., Automated Forest Classification and Inventory in the Eldorado National Forest, U.S. Department of Agriculture Forest Service Report, 1983.

TABLE 1 ADAM RIVER - MINIMUM REGION SIZES

Class	Minimum Size (Pixels)	Color Code
Hemlock Clearcut	100	Dark Green
Hemlock Mature Seral	100	Olive Green
Forrested Rock	100	Dark Brown
Red Alder	50	Bright Green
Young Hemlock		
Clearcut	250	Bright Yellow
Huckleberry Clearcut	100	Bright Orange
Fireweed Clearcut	100	Medium Red
Recent Clearcut	100	Bright Red
Rock	50	Medium Grey
River Bar	25	Light Blue
Water	15	Dark Blue
Snow	100	White

TABLE 2 ADAM RIVER - SIMILARITY MATRIX

To From											
	Hemcc	Hemms	Forrock	Red Alder	Young Hemcc	Huckcc	Fweedcc	Reccut	Rock	River Bar	Water Snow
Hemcc	5	4	4	3	2	1	1	1	1	0	0
Hemms	4	5	4	3	2	1	1	1	1	0	0
Forrock	4	4	5	3	2	1	1	1	1	0	0
Red Alder	4	4	4	5	2	3	3	2	1	0	0
Young Hemcc	2	3	2	1	5	4	4	3	1	0	0
Huckcc	2	2	2	2	4	5	4	3	1	0	0
Fweedcc	2	2	2	2	4	4	5	3	1	0	0
Reccut	2	2	2	1	4	4	4	5	2	0	0
Rock	1	1	4	1	3	3	3	4	5	0	2
River Bar	0	0	1	2	3	3	3	4	4	5	0
Water	0	0	0	0	1	1	1	2	3	0	5
Snow	0	0	1	0	0	0	0	1	4	0	5

TABLE 3 ADAM RIVER - NUMBER OF POLYGONS

Processing	Number of Polygons
Maximum Likelihood Classification (MLC)	39,316
3-by-3 Majority Filtered MLC	8,123
5-by-5 Majority Filtered MLC	3,798
7-by-7 Majority Filtered MLC	2,254
9-by-9 Majority Filtered MLC	1,433
Contextually Smoothed MLC	502

TABLE 4 ADAM RIVER - CLASSIFICATION ACCURACY

Processing	Accuracy (%)
Maximum Likelihood Classification (MLC)	74.9
3-by-3 Majority Filtered MLC	78.5
5-by-5 Majority Filtered MLC	80.6
7-by-7 Majority Filtered MLC	80.8
9-by-9 Majority Filtered MLC	79.1
Contextually Smoothed MLC	80.3

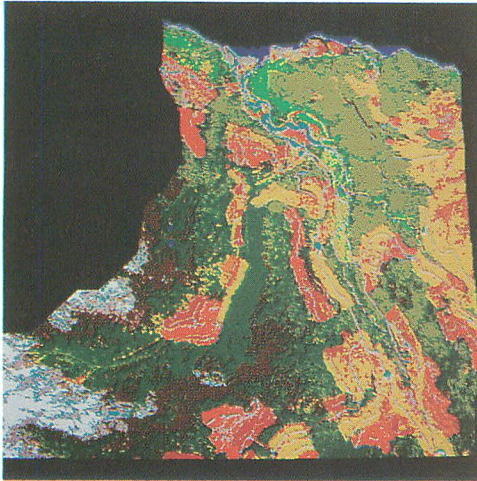


FIGURE 1 ADAM RIVER CLASSIFICATION

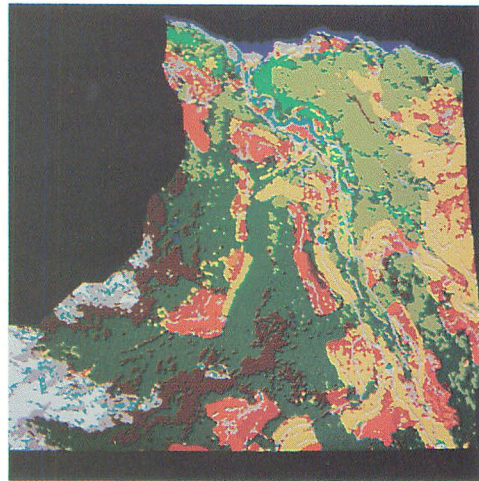


FIGURE 2 ADAM RIVER CLASSIFICATION
-3 BY 3 MAJORITY FILTERED

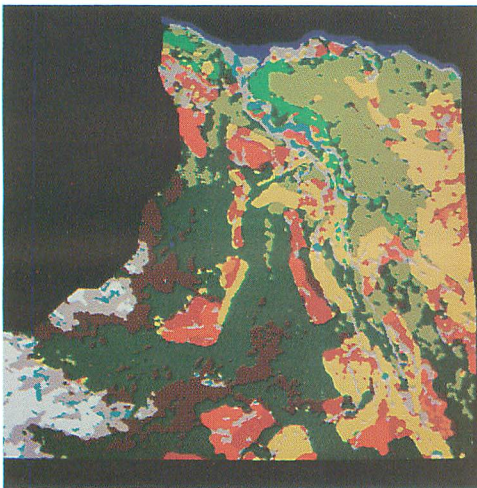


FIGURE 3 ADAM RIVER CLASSIFICATION
-5 BY 5 MAJORITY FILTERED

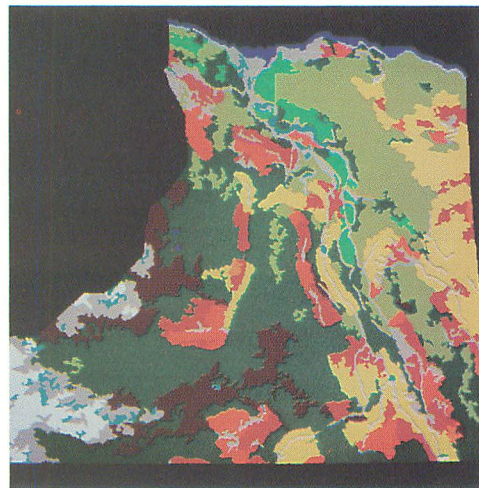


FIGURE 4 ADAM RIVER CLASSIFICATION
-CONTEXTUALLY SMOOTHED

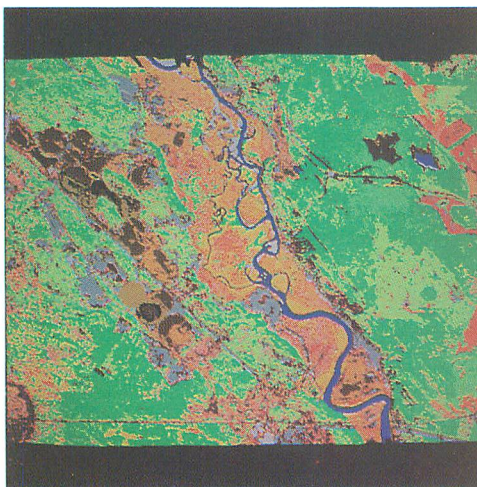


FIGURE 5 CRANBROOK CLASSIFICATION

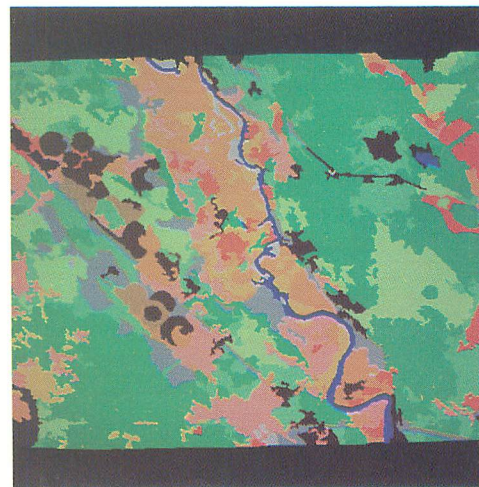


FIGURE 6 CRANBROOK CLASSIFICATION
-CONTEXTUALLY SMOOTHED

**"PRINCIPE DE CODAGE VISUEL DE LA COULEUR APPLIQUE
A DES IMAGES SATELLITAIRES"**

M-J. LEFEVRE-FONOLLOSA

Ingénieur - Division Traitement de l'Image

H. CRUCHANT

Technicien - Division Traitement de l'Image

CENTRE NATIONAL D'ETUDES SPATIALES

18, Avenue Edouard Belin

31055 TOULOUSE CEDEX

FRANCE

ABSTRACT

In remote sensing, color is used essentially as a means of enhancing the results of image processing. In recent months, we have investigated the use of color as a novel means of processing the spatial information content of image data.

Recent advances in the field of color vision by E.H. LAND (1978), T. WIESEL and D. HUBEL (1977) and S. ZEKI (1983), reveal that the visual system of higher mammals can be schematically divided into three successive segments, namely the optics (or eye, performing the "sensor" function), the neurobiological segment (from retina to cortex, performing the "encoding" function), and the cerebral segment (performing the "recognition and interpretation" function). Only the first segment is sensitive to electromagnetic radiation. The other two segments are conceptual and interactive.

Apart from certain other advantages, the processing performed by the three successive segments enables Man to recognize the color of an object irrespective of variations in the light illuminating that subject and in spite of the fact that such variation necessarily results in significant changes in the spectral composition of the reflected radiation.

We have attempted to use these characteristics of the human visual system in a novel approach to the processing of remote sensing imagery. Specifically, we use three channels of a spatial radiometer to simulate the "sensor" function of the human eye while using computer processing to simulate the function performed by the second segment of the system.

When an image is subjected to this type of simulation-processing, the result is three new-images termed the "color-coded image", the "lighting-coded image" and the "color-quantity-coded image".

The paper concludes with comments of this approach and its prospects with suitable reference to examples based on TM and (simulated) SPOT data.

RESUME

La couleur, en télédétection, est surtout utilisée comme un moyen de mise en valeur et de présentation des résultats. Durant ces derniers mois, nous avons envisagé la couleur comme un moyen original de traitement de l'information spatiale.

Les découvertes récentes sur la vision des couleurs (E.H. LAND, 1978 ; T. WIESEL et D. HUBEL, 1977 ; S. ZEKI, 1983), montrent que le système visuel chez les mammifères supérieurs se décompose schématiquement en trois segments successifs : optique (l'oeil -fonction "capteur"), neurobiologique (de la rétine au cortex -fonction "codage") et cérébral (fonction "cognitive et interprétative"). Seul, le premier est sollicité par le rayonnement électromagnétique, les deux autres étant conceptuels et interactifs.

Entre autre performance, ces traitements successifs permettent à l'Homme de reconnaître le couleur d'un objet quelle que soit la variation de l'éclairement qui le baigne, bien que cette variation entraîne nécessairement une sensible modification du rayonnement spectral réfléchi par cet objet.

Nous avons tenté de mettre en application cette caractéristique visuelle dans le traitement des images de télédétection : à partir de trois canaux d'un radiomètre spatial, lui-même considéré comme remplissant la fonction "capteur" de l'oeil, nous simulons par traitement informatique le second segment, "codage" des couleurs.

Les trois images nouvelles ainsi créées s'appellent : "color coded image", "lighting-coded image" et "color-quantity-coded image".

L'intérêt de cette approche est commenté à partir d'exemples tirés de données Thematic-Mapper et SPOT.

Depuis quelques années en matière d'image, l'effort général technique et scientifique a été considérable et s'est porté particulièrement sur la couleur. Nous n'envisageons plus la couleur comme un simple mélange trichrome, mais comme un signal spécifique traité à chaque étape de la chaîne visuelle. Dans l'application de ces connaissances en traitement de l'image de télédétection, il a fallu préalablement développer un système qui permette de connaître à coup sûr la couleur résultante sur n'importe quel reconstituteur ; nous citons deux exemples significatifs. D'autre part, nous utilisons l'une des particularités de la vision à décomposer l'image rétinienne en une image de couleur et une image d'éclairement pour l'appliquer à une vue de télédétection d'une forêt inégalement éclairée.

Entre la rétine et le cortex, le système visuel peut, schématiquement, se décomposer en trois segments successifs :

- le premier segment servant à **capter** l'information de couleur (cônes sensibles aux courtes, moyennes et grandes longueurs d'ondes), puis à **transmettre** cette information aux aires concernées du cortex par un signal modifié : un canal achromatique et deux canaux antagonistes chromatiques ;

- le second segment **code** la couleur dans la couche V4 de l'aire 17 suivant la réponse de deux types de cellules : les cellules WL et WLO qui sont sensibles à la composition de l'éclairement et les cellules CO qui sont sensibles à toute variation de couleur. Cette forme parallèle de codage de l'éclairement et de la couleur a l'avantage de nous offrir une stabilité des couleurs face aux éclaircissements divers ;

- le troisième segment, moins bien connu, est celui de l'interaction avec les autres sens, celui de l'influence de la culture, etc... Il nous permet d'**interpréter** la couleur.

Dans un premier temps, nous nous sommes intéressés aux systèmes de représentations de la couleur nécessaires pour l'utilisation des reconstituteurs. Mais, comme notre but est surtout la compréhension de la vision appliquée à l'image en général puis à l'image de télédétection en particulier, on a choisi un système physico-perceptif comme le modèle cylindrique de JUDD. Dans la pratique, on calibre les reconstituteurs de telle manière qu'ils soient en relation identifiée au modèle par une référence à des mires colorées organisées comme le préconise MÜNSELL.

L'image colorée de télédétection est semblable à toute autre image en tant que telle ; elle diffère cependant des autres par son contenu informatif. Pour pouvoir retrouver puis extraire l'information de cette image, il est évidemment nécessaire de lui appliquer un "traitement" ; ceci se fait en deux étapes : premièrement, un prétraitement qui rendra le signal exploitable, puis, deuxièmement, un traitement proprement dit qui sera spécifique de l'information résiduelle recherchée. L'information originale contenue peut rester soit in-situ comme dans le cas de la composition colorée, soit être tirée de son contexte et restituée graphiquement comme dans le cas d'une classification.

Dans le cas d'une composition colorée, notre principe a priori étant de respecter le plus possible la donnée radiométrique, nous allons donc nous attacher à identifier puis à sauvegarder les relations existantes entre chacun des invariants. Dans la coloration d'une telle image, nous tiendrons compte de la lisibilité de l'oeil, c'est-à-dire qu'à une différence de signal radiométrique doit correspondre une différence perçue de couleur, proportionnellement. Exemple : nous avons, dans ce but, appliqué les courbes de sensibilité de l'oeil au modèle de restitution choisi.

Dans un autre exemple, nous désirons, au contraire, extraire l'invariant informatif de son contexte. Notre principe sera ici de respecter le sens relationnel existant entre l'objet et l'interprète. Nos travaux nous ont montré l'importance d'une sémiologie de la couleur (à développer) dans le cas d'une coloration de classification, ceci, afin que l'interprète ait un meilleur accès aux données qu'il recherche.

Nous insistons donc dans ces deux cas, sur l'importance de la maîtrise de la couleur, de la connaissance des modèles de représentations colorées, des caractéristiques du système visuel et de la connaissance de la culture professionnelle de l'interprète.

La découverte de fonctions de **codage** du deuxième segment cervical (rétine-cortex) est récente (HUBEL et WIESEL). Elle permet de repousser les limites de la connaissance vers le troisième segment (intracervical "brain"). En quoi cette propriété de la chaîne visuelle peut-elle nous servir en matière de télédétection ? Dans ce domaine, nous possédons plusieurs bandes passantes (canaux) qui captent les informations spectrales d'un paysage terrestre. Si nous nous limitons à trois canaux, ce qui est le cas de SPOT, nous pouvons essayer de modéliser cette fonction de codage du signal spectral, c'est-à-dire de transformer les valeurs spectrales en deux termes qui codent indépendamment la couleur intrinsèque des objets et leur éclaircissement. En d'autres termes, il nous apparaît comme une application importante de la couleur de procéder à une fonction de réduction et de synthèse de l'information. Nous inspirant des découvertes du neurophysiologiste Britannique Sémir ZEKI, trois informations spectrales aboutiront à deux informations indépendantes.

1°) Nous créons une "color coded image", sensible aux différences de couleurs, quel que soit l'éclairement, assimilable grossièrement aux réponses des cellules CO (Color Coded cells) de ZEKI ; ceci est par ailleurs en accord avec les expériences de E.H. LAND concernant la stabilité de la perception des couleurs dans des environnements différents (Mondrians et Rétinex).

2°) Nous avons également créé deux images de quantification de la couleur assimilables grossièrement aux réponses des cellules WL et WLO (Wavelength cells) de ZEKI. C'est-à-dire sensibles à la composition de longueur d'onde et à l'éclairement :

- "lighting-coded image" égale à la quantité de flux total (décomposé en flux de couleur et flux de

blanc) liée à la composition de la signature spectrale et indicatrice de l'éclairement.

- "color-quantity-coded image" qui est la proportion du flux de couleur par rapport au flux total. C'est une variable indicatrice de la quantité de couleur indépendamment de la nature de la couleur elle-même.

Un algorithme simple nous permet de visualiser séparément ces trois images comme pourrait peut-être le faire un cerveau humain amputé de la fonction d'intégration. Nous décrivons les effets d'un tel traitement pour deux types de paysages : un paysage d'estran (Ile de Noirmoutier, Vendée, FRANCE) et un paysage forestier sur une topographie accidentée (Montagne-Noire, Lauragais, FRANCE).

L'analyse de ces travaux, qui reste d'ailleurs à confirmer, montre que cette approche psychosensorielle du traitement de l'image en télédétection offre à l'interprète un outil de discrimination nouveau et original.

En prolongement de cette étude, il nous paraît utile de réfléchir, d'une part, sur les problèmes calculatoires de la vision des couleurs et sur leur application dans le domaine de la vision par ordinateur ; d'autre part, et à condition que le langage intracervical Brain soit mieux connu, sur de nouvelles méthodes de photo-interprétation.

A File Organization Scheme for Polygon Data

Chung Hee Hwang & Wayne A. Davis

Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1

Abstract¹

This paper presents a file organization scheme for representing polygon data by a quadtree. The proposed scheme is an adaptive cell method that is based on *extendible hashing* and *interpolation-based index maintenance*. It aims, on the average, to locate the record associated with a given key with one disk access (or at most two), maintaining a high storage utilization ratio. It also aims to process range search and set operations efficiently. The dynamic file organization capabilities of the scheme and the algorithm for range search are described.

1. Introduction

Given a sequence of k points $P_i = (x_i, y_i)$, for $1 \leq i \leq k$, in a plane, a polygon with vertices P_i is the sequence of line segments, called edges, $P_1P_2, P_2P_3, \dots, P_kP_1$. If these k edges do not have any intersection points, the polygon is simple. A simple polygon divides the plane into two distinct regions. The interior of a simple polygon is called a *polygonal region*. A *complex polygonal region* is a polygonal region which is allowed to have one or more holes in it. Hereafter, a complex polygonal region is called a *polygon*. A *polygon network* for a study area is a set of disjoint polygons overlapping the study area such that the set of polygons yields a total partition of the study area. Each polygon in a polygon network has a unique name.

Although polygon networks have been traditionally represented in vector format, recently quadtree encoding of a polygon network has received increased attention. The quadtree [7] is a dynamic data structure developed to reduce the storage requirement of raster representation by aggregating homogeneous cells. Nevertheless, as the original quadtree concept was based on the assumption that quadtrees were resident in main memory, quadtree structures may not be directly applicable to data resident in external memory. For example, the need to follow pointers may lead to a larger number of page faults than are acceptable in an interactive environment.

In an effort to overcome the frequent page fault problem, there have been studies to represent a quadtree as a linear quadtree [5] and use a B-tree file structure in organizing the data [1,8]. While the B-tree organization of a linear quadtree is a significant improvement over the original quadtree organization in the expected number of disk accesses for single record retrieval, the absence of a localization property is a primary disadvantage. A query usually requires the whole file to be retrieved even though the query can be answered with

information from a small part of the file. For example, range search is very awkward and set operations are not efficient because there is no implied connection between data buckets in physical storage and regions in the search space. Furthermore, the B-tree organization of quadtree encoded data still needs several disk accesses to retrieve each record because it is essentially a tree that is accessible with $O(\log n)$ I/O operations, where n is the number of records in the file.

From this perspective, a file organization scheme is developed for polygon networks encoded as a linear quadtree with an aim to locate the record associated with a given key with an average of one disk access (or at most two), maintaining a high storage utilization ratio. Furthermore, the following types of spatial queries are to be supported efficiently: the point-in-polygon query, range search and set operations such as polygon union or intersection and polygon overlay. The scheme is an adaptive cell method and it is based on extendible hashing [4] and interpolation hashing [2], a k -dimensional generalization of linear hashing [6].

2. Definitions and Notation

Let the study area, $U = [0, 2^n]^2$, be an image of $2^n \times 2^n$ unit square pixels that intersects a polygon network, and let each of the pixels have a polygon name (hereafter called *color*) associated with it. Furthermore, let the polygon network on U be represented by a region quadtree. To yield an arbitrary but consistent total ordering among the blocks of a quadtree, the following hash function is introduced:

Definition 1. Let $(x, y) \in U = [0, 2^n]^2$ be the x and y coordinates of the lower-left corner of a block of a quadtree defined on U and have the following binary representation:

$$x = \sum a_i 2^i, \text{ and } y = \sum b_i 2^i, \text{ for } 0 \leq i \leq n-1,$$

where $a_i, b_i \in \{0, 1\}$. Then, an *order preserving hash function*, s , that maps (x, y) onto the key space of $[0, 4^n]$ is defined by:

$$s(x, y) = \sum (a_i 2^{2i+1} + b_i 2^{2i}), \text{ for } 0 \leq i \leq n-1.$$

Notice that the key produced for each of the blocks in this manner is essentially the same as the locational code of the block in linear quadtree encoding [5]. Now, a file that represents a polygon network by a quadtree is defined:

Definition 2. A file F representing a polygon network for the study area U by a quadtree is the set,

$$F = \{(K(L), S(L), C(L)) : L \in R\},$$

where R is the set of all leaf nodes (blocks) of a region quadtree on U ,

$K(L)$ is the key of L produced by s ,
 $S(L)$ is the size, or alternatively the level, of L , and
 $C(L)$ is the color of the pixels intersecting L .

¹This research was supported in part by Grant NSERC A7634.

In order to structure the file *F* using an adaptive cell method, the study area is partitioned into a set of blocks and/or subblocks which are defined in the following.

Definition 3. A block of depth d , $0 \leq d \leq \max d \leq 2n$, where $\max d$ is the predefined maximum depth of a block partition, is a rectangular region in the study area with a standard shape and a standard location that are the same as those of a region produced by recursively halving the study area d times with lines alternately perpendicular to the x and y axes. A block with its depth equal to $\max d$ is called a *minimal block*.

Definition 4. A subblock of depth d , where $\max d < d \leq 2n$, is a rectangular region in the study area with a shape and a location that are the same as those of a region produced by recursively halving the study area d times with lines alternately perpendicular to the x and y axes. Within each *minimal block* there exist at most two different depths of subblocks, i.e., d' and d'' such that $d'' = d' + 1$.

Throughout this paper, $\max d$ will be used to denote the predefined maximum depth of a block partition. The following definition is useful for defining an adaptive cell method.

Definition 5. A *fixed data bucket* is a bucket which contains no more than a predefined number of records, b , and an *expandable data bucket* is a bucket which may contain more than b records by attaching one or more overflow fields to it.

An adaptive cell method is now defined that organizes the leaf nodes of a region quadtree into a file.

Definition 6. An *adaptive cell method* of organizing a file *F* is an abstract data type which:

- (1) guarantees that, for every cell $G1$ and $G2$ and for every record $L1 \in F \cap G1$ and $L2 \in F \cap G2$, $\text{key}(L1) < \text{key}(L2)$ if $\text{index}(G1) < \text{index}(G2)$,
- (2) guarantees that, for every subblock $G1'$ and $G2'$ of a minimal block G and for every record $L1' \in F \cap G1'$ and $L2' \in F \cap G2'$, $\text{key}(L1') < \text{key}(L2')$ if $\text{index}(G1') < \text{index}(G2')$, and
- (3) asserts that every block of depth d has exclusively one fixed data bucket associated with it if $d < \max d$; otherwise (the case of a minimal block), it has associated with it either a single fixed bucket exclusively or two or more expandable buckets that are contiguously located in physical memory such that:
 - a) each expandable bucket is exclusively associated with exactly one subblock of the minimal block, and
 - b) the overall *load factor*, i.e., the ratio of the number of existing records to the number of slots available, of these expandable buckets is within some predefined range.

It is understood from Definition 6 that every data bucket is associated with a block or a subblock in the study area. Consequently, each data bucket has associated with it a depth which is equal to the depth of the block or the subblock it corresponds to. For an illustration of the concept of Definition 6, consider the polygon network in Fig. 1. Let the predefined maximum depth of a block partition $\max d = 4$ ($\max d$ is normally small so that the directory may be stored in main memory), the capacity of a data bucket $b = 5$, the capacity of an overflow field $b' = 3$, and the lower and upper limits of the load factor be 0.40 and 0.75, respectively. Then, for the image of Fig. 1a, the proposed scheme produces a partition as shown in Fig. 1b.

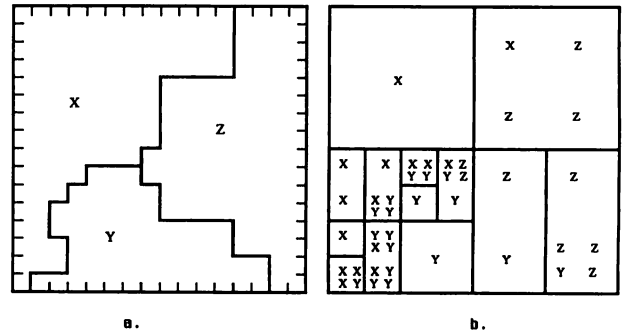


Fig. 1. Image of a Polygon Network and Partition of the Data Space.

3. Mapping between Regions and Data Buckets

Ordinarily the set of records in *F* is distributed over a number of data buckets, and each data bucket has associated with it a block or a subblock in the study area. The mapping between blocks and data buckets is achieved by a directory. A directory is a set of elements, each of which corresponds to a cell of size 2^{2n-d} , where d is the maximum of the depths of the existing blocks. Thus, a directory has associated with it a depth whose value is the same as d .

Each element of a directory has a pointer to a data bucket or a set of buckets which contains records describing the quadtree leaf nodes that intersect the corresponding cell in the study area. At depth d of a directory, there are altogether 2^d pointers, indexed from 0 to $2^d - 1$, which are not necessarily unique. The pointers of a directory are indexed in such a manner that a data bucket or a set of data buckets pointed to by a pointer with an index i contains all the records whose keys are prefixed with bits that are identical to the binary representation of i . That is, a data bucket or a set of data buckets pointed to by pointer 0 contains all the keys that start with d consecutive "0" bits, a data bucket pointed to by pointer 1 contains all the keys that start with $d - 1$ consecutive "0" bits followed by a "1" bit, and so on. Thus, the pointer i is guaranteed to find all the keys whose first d bits agree with the binary representation of i .

This indexing scheme is in fact equivalent to a Morton sequence [7] and naturally satisfies the first and second requirements of Definition 6. Fig. 2 illustrates the correspondence between the regions in the study area and directory elements (indexes are shown both in decimal and binary). Note that when the depth of a directory is odd, each pair of buddies at the deepest level are numbered consecutively from left to right, e.g., the pair 0 and 1 and the pair 2 and 3 in Fig. 2a.

a. Depth 3				b. Depth 4			
[2] 010	[3] 011	[6] 110	[7] 111	[5] 0101	[7] 0111	[13] 1101	[15] 1111
				[4] 0100	[6] 0110	[12] 1100	[14] 1110
				[1] 0001	[3] 0011	[9] 1001	[11] 1011
[0] 000	[1] 001	[4] 100	[5] 101	[0] 0000	[2] 0010	[8] 1000	[10] 1010

Fig. 2. Correspondence Between Regions & Directory Elements

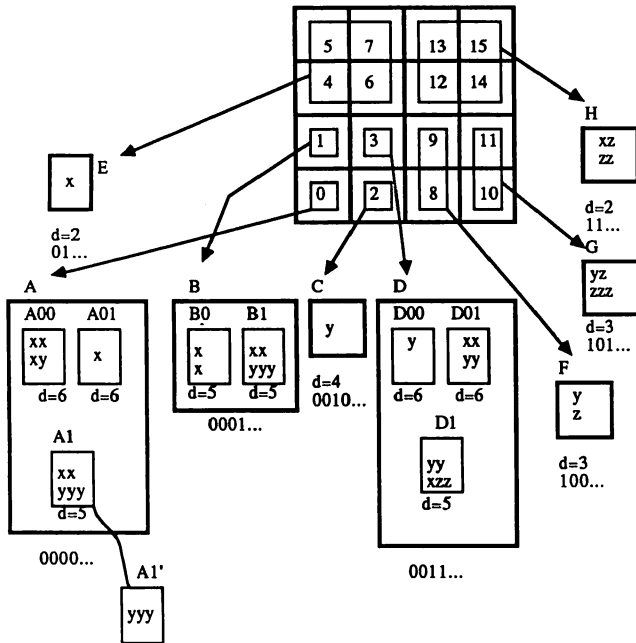


Fig. 3. Correspondence Between Blocks & Data Buckets.

The mapping between directory elements and data buckets, or sets of data buckets, is many-to-one. Fig. 3 shows the directory configuration corresponding to the partition shown in Fig. 1b. In Fig. 3, buckets C, E, F, G and H are fixed buckets, and buckets A00, A01, A1, B0, B1, D00, D01, and D1 are expandable buckets (bucket A1 has an overflow field attached to it). Note that all the records contained in a data bucket of depth d have the same bit pattern in their first d bits. Thus, bucket F, whose depth is 3, consists of the records whose keys start with "100", while bucket A00 whose depth is 6 contains all the records whose keys start with "000000". Also, notice that buckets A00, A01 and A1 contain 13 records in total while their capacity is 18. Thus, the load factor of these three expandable buckets is $13/18 = 0.72$, which is within the predefined range. The directory has 2^4 pointers because the largest of the depths of existing blocks is 4 which is the same as $maxd$. Note that pointer 0 points to a set of buckets (A00, A01 and A1) which contain all the records that start with "0000". The correspondence between subblocks and expandable data buckets, however, is not shown in the directory. That is, the corresponding pointer in the directory points to the starting address of a set of buckets that are physically located together, but it does not specify the correspondence between each of the subblocks and data buckets.

How the mapping between subblocks and data buckets are achieved will now be shown. Fig. 4 shows examples of a subblock partition of a minimal block. The subblocks of a minimal block are indexed in a similar manner as the cells corresponding to directory elements are indexed. In fact, when all the subblocks are the same size, they are indexed in the exactly same manner. Examples are shown in Figs. 4a and 4c. However, when there exist two different sizes of subblocks, the larger subblocks have two candidates for their index. In that case, the smaller of the two is selected for the index of the subblock as in Figs. 4b and 4d. As a result, when subblocks are of different sizes, the indexes of subblocks are not continuous.

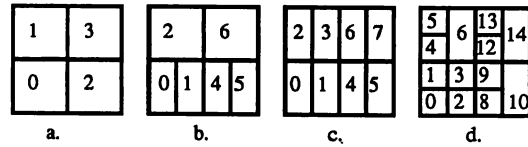
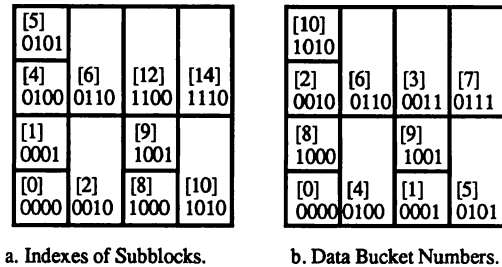


Fig. 4. Subblocks & Their Indexes.

Every subblock has an expandable data bucket associated with it. The keys of the records contained in a subblock of depth d have the same bit pattern in their leftmost d bit places. More explicitly, their leftmost $maxd$ bits agree with the index of the minimal block the subblock belongs to, and the next $(d - maxd)$ bits agree with the index of the subblock itself. Now, the mapping between subblocks and buckets is achieved by numbering each of the buckets that belongs to the same minimal block in a specific way as follows. Let a data bucket D be associated with a subblock whose index is i . Furthermore, let the maximum depth of existing subblocks be d . Then, i can be represented by a bit string S which is $(d - maxd)$ bits long. Next, let k be a number represented by a bit string S' which is the reversed bit string of S . Then, k is the bucket number of D , i.e., D is the $(k + 1)st$ of the set of buckets that are contiguously located. The proof of this "reversed bit pattern" relation between i and k can be easily shown by induction (see [2] for a formal proof). Fig. 5 illustrates the correspondence between data buckets and subblocks.



a. Indexes of Subblocks.

b. Data Bucket Numbers.

Fig. 5. Correspondence Between Subblocks & Data Buckets.

4. Dynamic Nature of the Scheme

The proposed file organization scheme allows a file structure to adapt its shape automatically to the nature of the data to be stored, i.e., the amount and the distribution pattern. The adaptability of the scheme is obtained mainly by a dynamic partition of the data space, which is implemented by *splitting* and *merging* mechanisms. In this section the merging mechanism is briefly described. See [3] for details of the dynamic file organization technique.

As more and more data is inserted in a file, data buckets overflow and this results in splitting of buckets. There are four kinds of splits possible. The first type of split occurs when a record is assigned to a data bucket that is full and pointed to by more than one pointer of the directory. In this case, the overflow bucket is split to resolve the collision, and the pointers in the directory are adjusted to reflect this split.

The second type of split arises when the overflow bucket is pointed to by a single pointer, and the directory has not reached its maximum yet. Then, in addition to a data bucket split, refinement of the cell partition in the data space is required as well as a directory doubling. A directory doubling involves copying of the entire directory in such a manner that the old contents of element i , for $i = 0, 1, \dots, 2^d - 1$, where d is the old value of directory depth, is copied into elements $2i$ and $2i+1$.

The third case occurs when the depth of the directory has reached its maximum already. Then, the overflow bucket is split into two expandable buckets, numbered 0 and 1, bucket 1 being physically allocated after bucket 0. This is called a *linear* bucket split.

The fourth type of split occurs when a record is assigned to an expandable bucket, and the load factor exceeds the upper limit as a result of insertion. Suppose a record is assigned to a data bucket of depth greater than d . Then, the record is first inserted into the bucket, or if necessary, into its overflow field, and the overall load factor of the set of buckets that are associated with the same minimal block is calculated and checked against the predefined range. If the load factor exceeds the upper limit, a *linear* bucket split is triggered, i.e., a new bucket is allocated at the end of the existing buckets of the set, and the bucket designated by the variable *next to split*, explained in the following, is split into two. If the load factor still exceeds the upper limit, the splitting process is repeated.

Similar to linear hashing [6], the following two variables are used to control linear bucket splits: j - *split level*, and p - *next to split*. The split level, j , indicates the level of linear splits within each minimal block. Initially, j is set to 0 for every minimal block but is increased as linear bucket splits are performed so that

$$j = \max \left(\begin{array}{l} \text{depths of all subblocks} \\ \text{within the minimal block} \end{array} \right) - \text{maxd.}$$

Next to split, p , points to the bucket which is to split next. It is initially 0 for every minimal block, but is increased by one as a linear bucket split occurs. However, at the end of each cycle of linear bucket splits, p is reset to 0. That is, during the first cycle of splits, bucket 0 is split; during the second cycle, first, bucket 0, and then bucket 1 is split; and during the k -th cycle, buckets are split in the order of $1, \dots, 2^{k-1} - 1$.

5. Range Search

This section describes how the proposed file scheme supports range search. Given two points, (x_1, y_1) and (x_2, y_2) , where $x_1 \leq x_2$ and $y_1 \leq y_2$, specifying a query rectangle, the proposed file scheme is able to retrieve every data bucket that contains the records describing the quadtree leaf nodes which overlap the query rectangle without retrieving any irrelevant data buckets. An algorithm for identifying the relevant data buckets for a given query rectangle is described using the example of Fig. 6. Suppose a directory has depth 4 which is the same as the predefined maximum depth. Suppose also that the shaded area in Fig. 6 is the query rectangle. Determination of the cells that intersect the query rectangle is done as follows:

- (1) Using **Algorithm Access** in Appendix B, determine the indexes of cells in which (x_1, y_1) and (x_2, y_2) are contained. In this example, they are 2 and 14.
- (2) Decompose the bit pattern of these indexes into their x and y components. Let the x -component of the higher index be the upper limit of x . Similarly, determine the lower and upper limits of y .

Lower index:	0010 (2)	xlow:	01 (1)
	xyxy	ylow:	00 (0)
Higher index:	1110 (14)	xhigh:	11 (3)
	xyxy	yhigh:	10 (2)

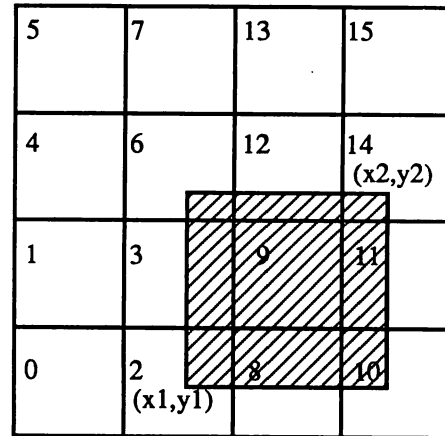


Fig. 6. Range Search.

- (3) For each of x and y , create a set of numbers which contains all the integers that are between the lower and upper limits inclusive.

$$x: \{01(1), 10(2), 11(3)\}$$

$$y: \{00(0), 01(1), 10(2)\}$$
- (4) Obtain a cross product of these sets, where each member of the resulting set is an integer produced by interleaving the x and y components. This set then indicates the cells that overlap the query rectangle. In this example, the following cells overlap the query rectangle: 2, 3, 6, 7, 10, 11, 12 and 14.

y \ x	01 (1)	10 (2)	11 (3)
00 (0)	0010 (2)	1000 (8)	1010 (10)
01 (1)	0011 (3)	1001 (9)	1011 (11)
10 (2)	0110 (6)	1100 (12)	1110 (14)

- (5) Next, suppose some of the cells have been further subdivided, e.g., cells 9 and 12. As for cell 9, every expandable bucket associated with it should be retrieved since the cell is completely contained in the query rectangle. Retrieval of these buckets can be done using **Algorithm SequenRetrieve** in [3]. As for cell 12, the subblocks that intersect the query rectangle should be determined in a similar manner as the cells intersecting the query rectangle. The bucket number corresponding to each relevant subblock is then obtained by reversing the bits of the subblock index. The detailed description of the algorithm is given in **Algorithm RangeSearch** in Appendix C.

6. Performance of the Proposed Scheme

In this section, the performance of the proposed file organization scheme is compared with other file organization schemes in terms of access efficiency for single record retrieval. The object for comparison is a B^+ -tree that has been proposed and implemented for representing a polygon network by a linear quadtree [1,8]. In addition, the EXCELL method is also used for comparison. Although the EXCELL method was originally used for representing a polygon network in vector format [9], the method is also useful for representing a polygon network by a quadtree.

The B⁺-tree maintains consistent performance both in storage and access time as its structure is not affected by the pattern of data distribution. Its buckets are more or less uniformly filled (each bucket being at least half full) even with non-uniformly distributed data, and every record in the file can be retrieved with $O(\log n)$ disk accesses, where n is the number of records in the file. In contrast, the proposed scheme and the EXCELL methods, i.e., cell methods in general, are very sensitive to the pattern of data distribution. With these methods, the best case occurs when data is distributed uniformly over the study area. Then, the proposed scheme requires one disk access to locate a record with a given key while the EXCELL method requires two disk accesses. The worst case occurs when data is distributed non-uniformly over the study area. With the EXCELL method, the directory will become large and unwieldy and the goal of two disk accesses for retrieving a record cannot be achieved. On the other hand, with the proposed scheme, there will be a long chain of overflow fields as well as many underflow buckets. Thus, both schemes may take $O(n)$ disk accesses for single record retrieval in the worst case.

The poor performance of a cell method is due to its extreme sensitivity to the existence of a random cluster of data. However, as a hybrid of extendible hashing and linear hashing, the proposed scheme allows its file structure to be considerably adapted to the nature of data. As the directory of the proposed scheme divides the study area into a coarse grid, any non-uniformity of data distribution affects the file structure only within a grid cell rather than over the entire study area. Furthermore, the probability of the worst case happening in practical data is expected to be exceedingly low. Since a worst case analysis does not provide meaningful conclusions, the performance of the proposed scheme has been simulated using a set of real data.

The scheme has been applied to a surficial geology map of the Wabamun area in Alberta, Canada (114°-115°W and 53.5°-54°N). Next, the same data have been used to estimate the performance of a B⁺-tree and the EXCELL method. It has been shown that the proposed scheme performs better than either a B⁺-tree or the EXCELL method in the expected number of disk accesses required to retrieve a record in a file, with a higher storage utilization ratio. See [3] for details. Although a formal proof cannot be given, it is conjectured that with the proposed scheme the expected number of disk accesses required for locating the record with a given key is constant irrespective of the file size, while that of B-trees or the (hierarchical) EXCELL method [11] grows logarithmically with the file size.

7. Conclusion

In most geometric databases, I/O operations are the bottleneck of their performance due to the large volume of data that should be handled. The proposed file organization scheme is an adaptive cell method which attempts to minimize the number of disk accesses in performing spatial queries. As a hybrid of interpolation hashing and extendible hashing, the proposed scheme combines the best features of both. First, the mapping between data buckets in physical storage and regions in the search space is interpolated rather than stored. Secondly, since a directory allows the search space to be divided into a coarse grid, any random cluster of data affects the file structure only within a grid cell rather than the entire file structure. Thirdly, a compromise between space and access time can be obtained by controlling the load factor.

Another important feature of the proposed scheme is that it handles spatial queries, range search in particular, efficiently by allowing a query to be decomposed into a set of subqueries within cell restrictions.

Experimental results with a set of real data show that the proposed scheme is superior to a B⁺-tree both in access efficiency and storage utilization. Additionally, the scheme is comparable to the EXCELL method which was originally proposed for representing a polygon network by vectors.

BIBLIOGRAPHY

1. Abel, D.J., "A B⁺-Tree Structure for Large Quadrees", *Computer Vision, Graphics, and Image Processing*, Vol. 27, pp. 19-31, 1984.
2. Burkhard, Walter A., "Interpolation-Based Index Maintenance", *BIT*, Vol. 23, pp. 274-294, 1983.
3. Davis, Wayne A. & Chung Hee Hwang, "File Organization Schemes for Geometric Data", TR 85-14, Dept. of Computing Science, Univ. of Alberta, 1985.
4. Fagin, R., J. Nievergelt, N. Pippenger & H.R. Strong, "Extendible Hashing - A Fast Access Method for Dynamic Files", *ACM TODS*, Vol. 4, pp. 315-344, Sep. 1979.
5. Gargantini, Irene, "An Effective Way to Represent Quadrees", *CACM*, Vol. 25, pp. 905-910, Dec. 1982.
6. Litwin, Witold, "Linear Hashing: A New Tool for File and Table Addressing" *Proc. 6th Int. Conf. Very Large Data Bases*, pp. 212-223, 1980.
7. Samet, Hanan, "The Quadtree and Related Hierarchical Data Structures", *Comput. Surveys*, Vol. 16, pp. 187-260, Jun. 1984.
8. Samet, H., A. Rosenfeld, C.A. Shaffer & R.E. Webber, "A Geographic Information System Using Quadrees", *Pattern Recognition*, Vol. 17, pp. 647-656, 1984.
9. Tamminen, Markku, "Efficient Spatial Access to a Data Base", *ACM-SIGMOD*, pp. 200-206, 1982.
10. Tamminen, Markku, "The Extendible Cell Method for Closest Point Problems", *BIT*, Vol. 22, pp. 27-41, 1982.
11. Tamminen, Markku, "Performance Analysis of Cell Based Geometric File Organizations", *Computer Vision, Graphics, and Image Processing*, Vol. 24, pp. 160-181, 1983.

APPENDIX

A. Data Structure

The file structure consists of a directory and data buckets. The directory has a *header* containing the depth of the directory followed by 2^d elements, where d is the depth of the directory. Each element of the directory is a 5-tuple of $\langle j, p, occ, over, ptr \rangle$, where j is the split level, p is the bucket to be split next, occ is the number of records contained in the data bucket (or set of expandable buckets), $over$ is the number of overflow fields employed, and ptr is the pointer to the data bucket (or set of expandable data buckets).

Each data bucket or overflow field contains a set of records, (K(L), S(L), C(L)). In addition to the set of records, each data bucket has a header that contains db , the bucket depth, and a pointer, ptr . If a bucket is an expandable one, ptr points to a chain of overflow fields attached to it; otherwise, it may be either ignored or used to point to the next bucket.

B. Algorithm Access

Input: file *F*, directory *dir* to *F*, and $(x,y) \in U$,
where $U = [0,2^n]^2$ is the study area.
Output: data bucket *D* which contains $(K(L),S(L),C(L))$ such
that $(x,y) \in L$. (The record may be in an overflow field
of *D*.)

Note: *dir*[*i*].*A* denotes field *A* of (*i*+1)*st* element of directory.

Step 1: *key* $\leftarrow s(x,y)$
Step 2: read *d*, depth of directory
Step 3: *i* $\leftarrow \lfloor (key / 2^{2n-d}) \rfloor$ #determine *dir* index#
Step 4: *loc* $\leftarrow dir[i].ptr$ #read pointer value#
Step 5: if *dir*[*i*].*j* = 0, goto Step 8
Step 6: #case of split level $\neq 0$ #
 a. if *dir*[*i*].*p* = 0, #every bucket split#
 1) #set subblock index with *j* bits of the *key*#
 isub $\leftarrow \lfloor (key \bmod 2^{2n-d}) / 2^{2n-d-j} \rfloor$,
 where *j* denotes *dir*[*i*].*j*
 2) #calculate bucket no.#
 bnum $\leftarrow \sum a_k 2^{m-1-k}$, for $0 \leq k \leq m-1$, where
 isub is $\sum a_k 2^k$, for $0 \leq k \leq m-1$
 b. otherwise
 1) #set subblock index with (*j*-1) bits of the *key*#
 isub $\leftarrow \lfloor (key \bmod 2^{2n-d}) / 2^{2n-d-j+1} \rfloor$,
 where *j* denotes *dir*[*i*].*j*
 2) #calculate bucket no.#
 bnum $\leftarrow \sum a_k 2^{m-1-k}$, for $0 \leq k \leq m-1$, where
 isub is $\sum a_k 2^k$, for $0 \leq k \leq m-1$
 3) #if bucket split, adjust bucket no.#
 if *bnum* < *dir*[*i*].*p* and (*d*+*j*)*th* bit of *key* = "1",
 bnum $\leftarrow bnum + 2^{j-1}$
Step 7: #calculate address of the bucket#
 loc $\leftarrow loc + bnum * unit-length$, where
 unit-length is the bucket size
Step 8: access bucket *D* at *loc* and exit.

C. Algorithm RangeSearch

Input: file *F*, directory *dir* to *F*, and $(x_1,y_1),(x_2,y_2) \in U$,
where $x_1 \leq x_2$ and $y_1 \leq y_2$.
Output: retrieve every data bucket whose associated block or
subblock in *U* intersects the query rectangle specified by
 (x_1,y_1) and (x_2,y_2) .

Step 1: *R* $\leftarrow \{ \}$ #initialize index set#
Step 2: read *d*, depth of directory
Step 3: #calculate keys and indexes#
 key1 $\leftarrow s(x_1,y_1)$; *i1* $\leftarrow \lfloor key1 / 2^{2n-d} \rfloor$
 key2 $\leftarrow s(x_2,y_2)$; *i2* $\leftarrow \lfloor key2 / 2^{2n-d} \rfloor$
Step 4: if *i1* = *i2*, #trivial case#
 R $\leftarrow \{i1\}$ and goto Step 12
Step 5: #determine limits of index#
 a. if *d* is even,
 1) *xlow* $\leftarrow \sum a_{2k+1} 2^k$, for $0 \leq k \leq \lfloor m/2 \rfloor$
 2) *xhigh* $\leftarrow \sum b_{2k+1} 2^k$, for $0 \leq k \leq \lfloor m/2 \rfloor$
 3) *ylo* $\leftarrow \sum a_{2k} 2^k$, for $0 \leq k \leq \lfloor m/2 \rfloor$
 4) *yhigh* $\leftarrow \sum b_{2k} 2^k$, for $0 \leq k \leq \lfloor m/2 \rfloor$
 b. otherwise,
 1) *xlow* $\leftarrow \sum a_{2k} 2^k$, for $0 \leq k \leq \lceil m/2 \rceil$
 2) *xhigh* $\leftarrow \sum b_{2k} 2^k$, for $0 \leq k \leq \lceil m/2 \rceil$
 3) *ylo* $\leftarrow \sum a_{2k+1} 2^k$, for $0 \leq k \leq \lfloor m/2 \rfloor$
 4) *yhigh* $\leftarrow \sum b_{2k+1} 2^k$, for $0 \leq k \leq \lfloor m/2 \rfloor$
 where *i1* (*i2*) is $\sum a_k 2^k$ ($\sum b_k 2^k$), for $0 \leq k \leq m-1$

Step 6: *ix* $\leftarrow xlow$ #initialize *x* index#
Step 7: *iy* $\leftarrow ylow$ #initialize *y* index#
Step 8: #compute index of relevant block#
 if *d* is even, *i* $\leftarrow shuffle(ix,iy)$;
 otherwise, *i* $\leftarrow shuffle(iy,ix)$, where
 $shuffle(V,W) = \sum (2v_k + w_k) 2^{2k}$, for $0 \leq k \leq m-1$
 given $V = \sum v_k 2^k$ and $W = \sum w_k 2^k$,
Step 9: *R* $\leftarrow RU \{i\}$ #store the index#
Step 10: #continue until *y* upper limit is reached#
 1) increase *iy* by 1
 2) if *iy* $\leq yhigh$, goto Step 8
Step 11: #continue until *x* upper limit is reached#
 1) increase *ix* by 1
 2) if *ix* $\leq xhigh$, goto Step 7
Step 12: *loc* $\leftarrow null$ #initialize#
Step 13: for each member *i* in *R*, perform
 a. #determine address of bucket#
 if *dir*[*i*].*ptr* $\neq loc$, *loc* $\leftarrow dir[i].ptr$;
 otherwise, goto Step 13.e
 b. if *dir*[*i*].*j* = 0,
 retrieve bucket at *loc* and goto Step 13.e
 c. #linear splits have occurred#
 if the block is totally contained in the query rectangle,
 retrieve every bucket belonging to the block using
 Algorithm SequenRetrieve and goto Step 13.e
 d. #the block is partially contained#
 1) *R'* $\leftarrow \{ \}$ #initialize subblock index set#
 2) let (x_1',y_1') and (x_2',y_2') , where $x_1' \leq x_2'$ and
 $y_1' \leq y_2'$, be the points specifying the rectangle
 which is the intersection of the current minimal block
 and the query rectangle
 3) compute *R'* in a similar manner to Steps 3-11
 Note: in Step 8, *d* should be substituted with *j*
 4) #compute the number of buckets#
 if *dir*[*i*].*p* = 0, *M* $\leftarrow 2^j$;
 otherwise, *M* $\leftarrow 2^{j-1+p}$
 5) for each member *isub* in *R'*,
 compute bucket number *bnum*;
 if *bnum* < *M*,
 calculate address of bucket and retrieve
 e. continue.

MATHEMATICAL MORPHOLOGY APPLIED TO RANGE IMAGE PROCESSING

Colin C. Archibald
Machine Vision International Ltd., 280 Albert St., Ottawa, Canada. K1P 5G8

Stanley R. Sternberg
Machine Vision International Corp., 325 E. Eisenhower Pway Ann Arbor, MI U.S.A. 48104

ABSTRACT

Methods of low level image processing have predominantly been derived by generalizing one dimensional signal processing methods to 2 dimensions. Although much progress has been made using Fourier domain analysis, it is still unintuitive and inflexible for many types of applications. This paper reviews mathematical morphology as a basis for low level image processing, and demonstrates that image domain transformations can be applied usefully to various types of images. Specifically, range images are processed using grey scale morphology to extract various features used in three dimensional analysis of a scene. This type of scene analysis has broad potential for applications in quality control and automated manufacturing.

RESUME

Le traitement primaire d'image a traditionnellement été inspiré des méthodes de traitement de signaux en les généralisant pour deux dimensions. Le traitement basé sur l'analyse dans le domaine de Fourier a permis d'important progrès mais défie l'intuition et demeure trop peu flexible pour nombre de problèmes.

Nous faisons ici un compte rendu des applications de la morphologie en tant que base d'analyse primaire de l'image en l'appliquant à diverses classes d'images. Plus précisément, des images tri-dimensionnelles sont traitées par la méthode morphologique en tons de gris pour extraire divers éléments propres à l'analyse de scènes tri-dimensionnelles.

Cette classe d'analyse ouvre un large éventail de possibilités dans les applications de contrôle de qualité et de fabrication assistée par ordinateur.

1. INTRODUCTION

The arrival of mathematical morphology for image processing has allowed North Americans to

reconsider their approach to image transformation. Transform domain techniques, like the Fourier transform, used to decompose an image into constituent spatial frequencies, grew out of classical one dimensional signal processing where the dominant analytical theme is linear transformation. Conventional image processing extended the subject base to two or more dimensions, but the paradigm remained one of filter design for removal of noise and enhancement of visual fidelity.

A morphological approach allows the designer of vision systems a ubiquitous tool to perform image transforms in the image domain, using the algebra of shapes. Although the morphological treatment of images has been studied for more than 50 years, its recent popularization is due mostly to Serra (1), and Sternberg (2) (3), the latter introducing greyscale morphology.

The use of mathematical morphology has overcome several obstacles in the development of industrial vision systems. Mathematical morphology is first of all a mathematics of image transformation and analysis, thus it forms the basis for a powerful language of image processing. This language is not only powerful, it is intuitive, and can be understood visually while developing image processing algorithms interactively. As a general purpose method of image transformation, all types of images can be processed regardless of the sensor used to collect the image. Satellite, microscope, x-ray, T.V., ultrasound, tactile array sensors, laser range finder images, etc., can all be transformed productively using mathematical morphology. This paper describes the basic operations of how to transform an image with a shape, calling on the readers intuition instead of the mathematical basis of the transformations which are exhaustively described in (1) and (4). We will demonstrate, using figures, transformations on binary and grey scale images, including images acquired using a laser range finder. It will be pointed out that these morphological transforms can be used for low level (data driven), and high level (model driven) aspects of industrial applications.

II. RANGE IMAGE PROCESSING

Currently range finding cameras are becoming commercially available. These devices use passive or active light sources to record two dimensional arrays of depth, i.e., the distance from the camera to the surface being imaged. Active light sources (often lasers) are used in two ways to collect depth information. Time of flight range finders are similar in concept to radar where the time required for the laser to be reflected from the scene is used as a measure of distance. Triangulation based range finders depend on the location of the reflected light to calculate the distance. A triangulation range finder developed in the Division of Electrical Engineering at the National Research Council was used in this research, and is described in (5). (A survey of range finding methods can be found in (6).) The elements in a range image are referred to as "surfels" or surface elements, and must be processed using algorithms designed specifically for this type of image. In some respects a range image is easier to analyze than intensity images because the information in this type of image is a function of only one scene property, namely depth. Pixels in intensity images, such as those from T.V. cameras, are dependent on reflectivity, lighting intensity and direction, colour, etc., making robust algorithms elusive for all but the most controlled environments.

Range image processing research has centered around extraction of information contained in edges and/or surfaces. Although it can be argued that one leads to the other, the various methods for extracting edge features differ significantly from region detection. Each method has virtues, depending on the ultimate goal of the research. Edge finding methods have been used in (8) (9) (10). Gil et al., (11) used registered range and intensity images to build a more complete edge map. Planar and/or quadratic surfaces are extracted in (7) (12) (13) (14). An interesting approach to collecting the maximum information from the image is to combine the edge and region detection methods. An operator for this purpose is described in (10). A more indepth survey of related literature can be found in (15). Generally, both edge and region finding methods have grown from extending and manipulating ideas used in intensity images. A critical assumption which can be made about range images has been ignored. The information in a range image is not predominantly in the edges or the regions, but in the three dimensional shape represented by the image data. Instead of processing range images with edge or region operators, we suggest that shapes should be used. These shapes exist in the same coordinate system as the image data making the concept of shape transformations easy to visualize. A mathematics of shape transformation

called mathematical morphology is described in the following section.

III. MATHEMATICAL MORPHOLOGY

Mathematical morphology is the algebraic treatment of shapes. A transformation based on a shape (sometimes referred to as a structuring element) is surprisingly easy to understand with the use of a simple visual example. There are two classical transformations that are most often used; erosion and dilation. Consider first the two dimensional, binary image application of these operations. What does it mean to erode and dilate a binary image with a shape? To illustrate the answer to this question we have inserted three figures of watch gears. Fig. 1 is the original binary image, Fig. 2 is a dilated image, and Fig. 3 is an eroded image. Notice the small disk in Fig. 1. This shape is used to dilate and erode the image into Figs 2 and 3 respectively. The disk is not a part of the image, but is placed in the figure to help the reader visualize the transformations. In the following textual description, Fig. 1 is the original image, and Fig. 2 is the resultant image. To perform a dilation of the original image, place the structuring element (in this case a disk) so that the centre point falls on a pixel i, j in the original image. If the value of the pixel i, j in the original image is 1, then each pixel which is covered by the structuring element will become 1 in the resultant image. This step is performed for all pixels in the image. The reader is encouraged to verify that this is indeed how Fig. 2 resulted from Fig. 1.

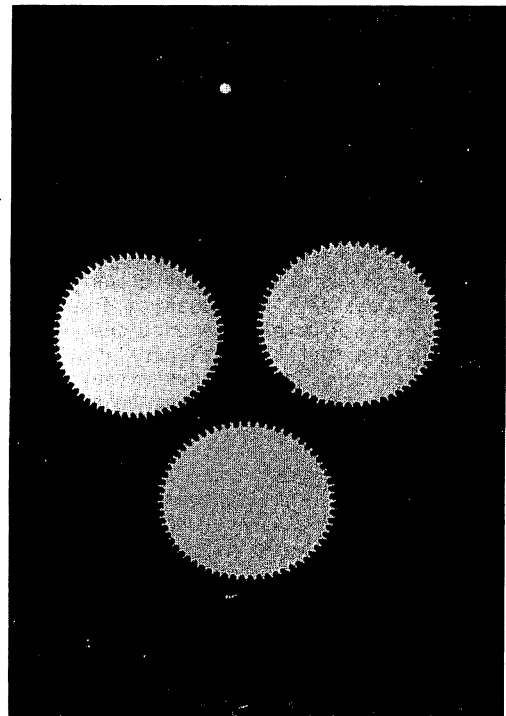


Fig. 1 Binary image of watch gears.

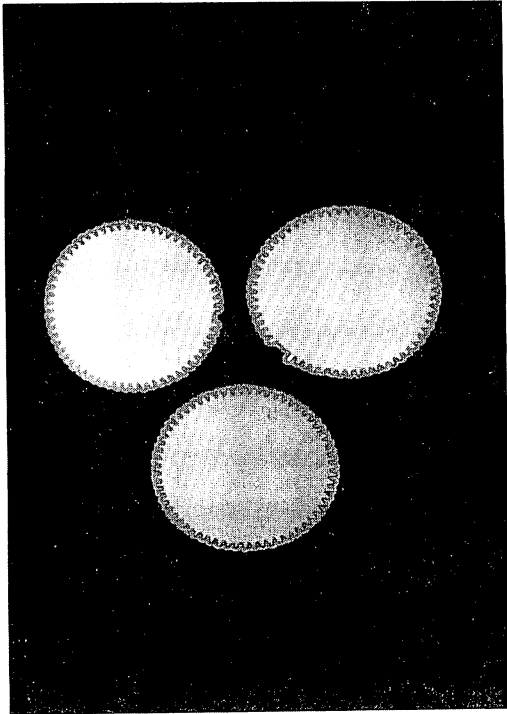


Fig. 2 Dilated watch gears.

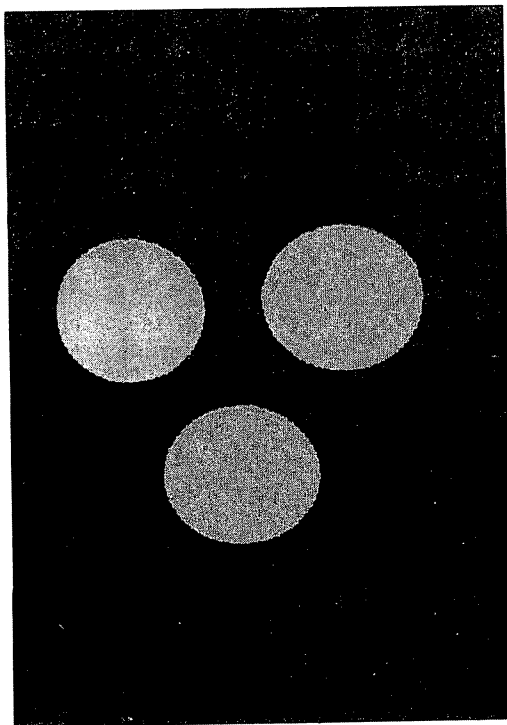


Fig. 3 Eroded watch gears.

The erosion transformation can be visualized in two ways. If the dilation is clearly understood, then the erosion transformation can be considered a dilation of the background, i.e., dilate the 0's in the image instead of the 1's. Notice that Fig. 3 is the result of eroding Fig. 1. with the same structuring element used previously. The second way to understand an erosion, is fundamentally the same as the description of the dilation. Place the structuring element at each pixel i, j in the original image. If the value of pixel i, j is 0 then, each pixel which is covered by the structuring element will become 0 in the resultant image.

The structuring element of a small disk was used in this explanation to demonstrate the utility of such a transform in an industrial application. In this case, a priori knowledge of the size of the gear and spacing of the gear teeth allowed the choice of the correct size structuring element which could be used to identify the broken teeth on the gears. This is an example of a model driven transform, where the model was a priori knowledge of the size and shape of the gear.

Next consider what it means to transform a greyscale image, either range or intensity, with a three dimensional shape such as a ball. To facilitate this description we point out that a grey scale image can be thought of as a function $f(x,y)$ on the points of Euclidean 2-space. In 3-space a grey scale image is a set of points $x,y, f(x,y)$ where $f(x,y)$ is the pixel value. In a range image $f(x,y) = z$ since the image is a three dimensional representation of the scene. This can be visualized as a thin, not necessarily continuous sheet. Fig. 4 is an oblique graphical representation of a range image, demonstrating how an image can be thought of as a sheet. (This image is further discussed in the following section.)

Greyscale erosions and dilations on these sheets are most often used in conjunction with each other. An erosion followed by a dilation is called an opening, and a dilation followed by an erosion is called a closing. The opening or closing of a sheet by a geometrical solid is a grey scale transformation which treats different portions of an image uniquely, depending on how well the local grey level topology is matched by the shape.

Fig. 5 illustrates a greyscale closing operation, using a ball. The ball is rolled over every pixel (surfel) on the image. Where the topology of the sheet prevents the ball from touching each pixel, the value of the pixel is raised to the surface of the ball. A greyscale opening is the mathematical dual. The ball is rolled on the underside of the sheet, and the

pixel values are lowered to touch the ball. (To see this turn Fig. 5 upside down.)

Fig. 6(a), 6(b), and 6(c) are images of a man's face, where (a) is the original image, (b) is closed using a ball, and (c) is the residue image, i.e., Fig. 6(a) - Fig. 6(b). Fig. 6(b) is a version of Fig.(a) with some information removed. The removed information is considered to be where the ball would not fit into the sheet that represents the original image. A pixelwise subtract of Fig. 6(a) - Fig.6(b) actually extracts this information and is shown in Fig. 6(c). This type of residue image is a common method of extracting useful information in industrial machine vision algorithms.

The power of these openings and closings is realized in the fact that these operations can be performed in real time using any shape and size of that shape that the application requires.

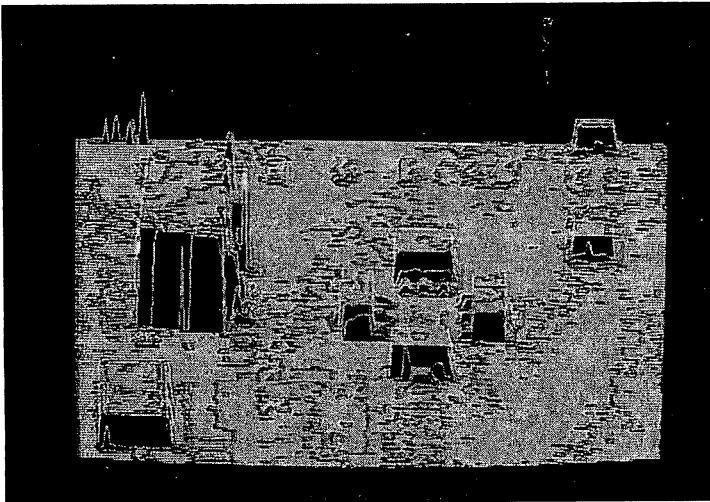


Fig. 4 Oblique graphical representation of a range image demonstrating how an image can be thought of a sheet.

IV. MORPHOLOGICAL PROCESSING OF RANGE IMAGES

Three dimensional information is explicit in a range image. Applying three dimensional operators to this data is demonstrated as a useful approach to transforming three dimensional data. Transforming data in the same coordinate system in which it is originally represented becomes feasible for real time applications with the advent of specialized hardware capable of 3-D morphology. The ability to choose the size and shape of the operator used to transform the image implies some a priori knowledge. Depending on the objective of the processing, this a priori knowledge may be a low level of characterization of

the image properties, such as signal to noise ratio, up to a complete 3-D model of what is expected in the scene. To illustrate this we examine the application of component placement verification on printed circuit boards. With automated component placement, it is necessary to inspect the boards visually to ensure that the components are present and properly aligned. Fig. 4 is a graphical representation of a range image that shows components placed on a printed circuit board. This image is shown in Fig. 7(a) displayed as intensity, i.e., the bright areas are nearer the viewer than the dark background. If the range imaging process were ideal, an exact 3-D representation of the scene would be contained in the data. There are some clearly defined reasons why this is not the case (5). The shadow effect, and specular reflectance, are inherent in this technology and cause errors in the data as can be seen in Fig. 4 and Fig. 7(a). It is often the case that the errors are considerably smaller than the objects or features of interest to the application. This presents the familiar problem of processing the image to remove the error that is relatively small, while preserving the information of interest. Typically this low level processing would be followed by model driven, higher level processing. A single morphological operation can accomplish both of these tasks.

Returning to the component inspection application; we know a priori the size of the smallest component expected in the scene. Thus, we can choose a structuring element that is known will fit inside the data representing the smallest component expected in the scene. Using this structuring element, a 3-D morphological closing is performed, which in effect removes all peaks in the data that are smaller than the smallest object of interest. Fig. 7(b) is the result of this closing operation. A threshold is used to create a binary image from this processed image. Fig. 7(c) is the binary image representing the location of the components. Well known connectivity analysis of this binary image quantifies the results which can be compared to the expected data.

This processing is so simple in concept that automatic image processing programming based on a priori knowledge becomes feasible for some highly structured environments. It is not difficult to generate the steps for the application described above, based on a simple data base representation of the printed circuit boards. It is less clear that this is useful for more loosely structured applications, but is worth considering one application at a time.

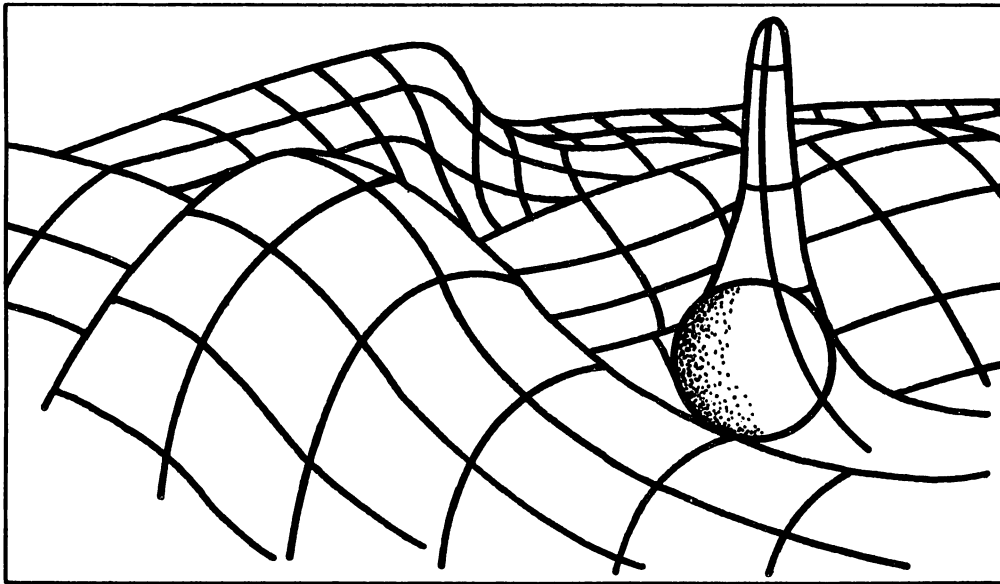


Fig. 5 The rolling ball algorithm illustrated graphically.



Fig. 6(a) A digital image of a man's face.



Fig. 6(b) The image closed using the rolling ball algorithm.



Fig. 6(c) Residual image, Fig 6(a) - Fig. 6(b)

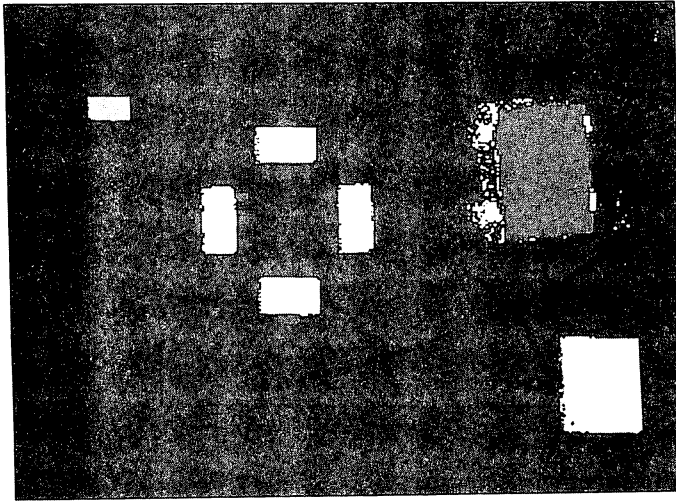


Fig. 7(a) Range image of a surface mount technology board, displayed as intensity. The bright areas are components, raised from the substrate.

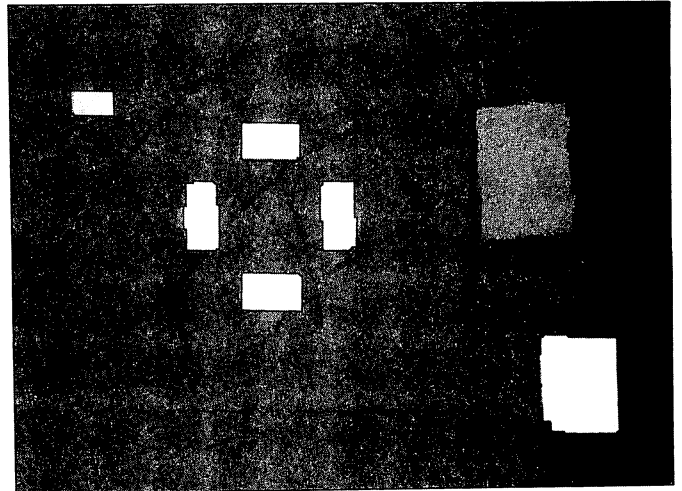


Fig. 7(b) The transformed image, using a morphological closing with a structuring element that is known to fit inside the smallest component.

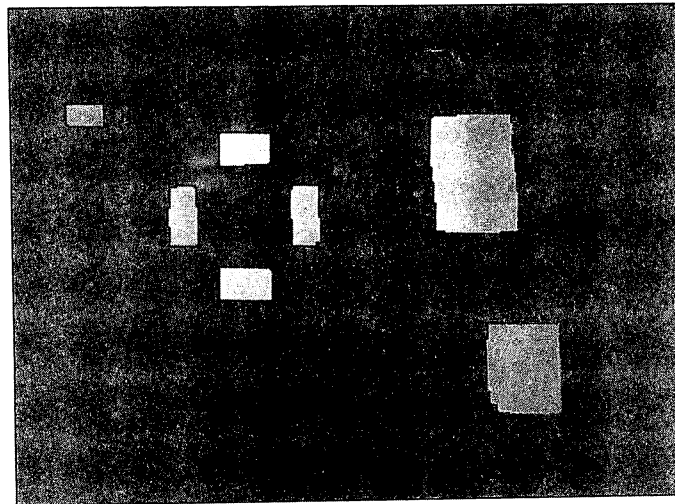


Fig. 7(c) A binary image resulting from thresholding Fig. 7(b).

V. CONCLUSION

Mathematical morphology can be used to transform images in both data driven and model driven processes. Choosing shapes (structuring elements) for data driven transforms is usually based on characteristics of the imaging device, and may accomplish such things as noise removal, edges effect removal, etc. In the model driven part of the algorithm, shapes are chosen based on a priori knowledge of what is expected in the scene. The morphological transforms are applied using these shapes to locate expected attributes in the image.

Morphological transformations are ideal for processing of range images. Industrial applications for verification vision are plentiful, and using the methods described in this paper it is possible to totally automate the inspection task with sufficient a priori knowledge combined with highly reliable range images containing explicit 3-D information. Although this scenario is the most appealing, there is a broad spectrum of applications varying on how much a priori knowledge is available, and the quality and type of images acquired. Mathematical morphology is sufficiently general to be used at all levels of image and scene analysis over this spectrum.

REFERENCES

- (1) Serra, J., *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.
- (2) Sternberg, S., *Industrial Morphology*, Proc. Interobot West, Long Beach Cal, Oct. 10-12, 1984.
- (3) Sternberg, S., *Industrial Computer Vision by Mathematical Morphology*, Proc. of the 13th International Symposium on Industrial Robots, Robot 7, Chicago, Illinois, April 18-20, 1983.
- (4) Sternberg, S., *Grayscale Morphology*, in *Algorithms in Mathematical Morphology*, Special issue of IEEE Computer, J.P. Serra and S.R. Sternberg, editors (in preparation).
- (5) Rioux, M., *Laser Range Finder Based on Synchronized Scanners*, Applied Optics, v. 23, no. 21, Nov. 1984.
- (6) Jarvis, R.A., *A Perspective on Range Finding Techniques for Computer Vision*, IEEE Transactions on Pattern Recognition and Machine Intelligence, Vol. PAMI-5, No. 2.
- (7) Archibald, C and Rioux, M., *WITNESS: A System for Object Recognition Using Range Images*, National Research Council of Canada, Technical Report ERB-986, 1985.
- (8) Nitzan, D., Brain, A., and Duda, R., *The measurement and Use of Registered Reflectance and Range Data in Scene Analysis*, Proc. of the IEEE, Vol. 65, No. 2.
- (9) Mitiche, A. and Aggarwal, J.K., *Detection of Edges Using Range Information*, IEEE PAMI, Vol. PAMI-5, No. 2, 1982.
- (10) Inokuchi, S., Nita T., Matsuda, F., and Sakurai, Y., *A Three Dimensional Edge-Region Operator for Range Pictures*, Proc. of the 6th International Conf. on Pattern Recognition, Munich, Germany, 1982.
- (11) Gil, B., Mitiche, A. and Aggarwal, J.K., *Experiments Combining Intensity and Range Edge Maps*, CVGIP21, 1983.
- (12) Duda, R.O., Nitzan and Barrett, P., *Use of Range and Reflectance Data to Find Planar Surface Regions*, IEEE PAMI, Vol. PAMI-1(3), 1979.
- (13) Hebert, M. and Ponce, J. *A New Method for Segmenting 3-D Scenes into Primitives*, Proc. of the 6th International Conf. on Pattern Recognition, Munich, Germany, 1982.
- (14) Henderson, T.C. and Bhanu, B., *Three Point Seed Method for Extraction of Planar Surfaces from Range Data*, Conference Record of the 1982 Workshop on Industrial
- (15) Besl, P.J. and Jain, R.C., *Three Dimensional Object Recognition*, ACM Computing Surveys, Vol. 17, No. 1, March 1985.

INCREMENTAL CONSTRUCTION OF 3-D MODELS FROM A SEQUENCE OF FRAMED VIEWS : MATCHING PARTIAL OBJECTS

Shun-en Xie and Thomas W. Calvert

LCCR, School of Computing Science, Simon Fraser University
Burnaby, B.C., V5A 1S6, Canada

ABSTRACT

In the future intelligent mobile robots will be called upon to play many important roles. In many realistic situations, the knowledge of the structure and placement of objects in an environment should be learned rather than built in. Thus the mobile robot must often construct 3-dimensional models for the objects by analysing sensed multiple views.

In this paper, we describe an approach to the incremental construction of 3-D body models in a practical office or warehouse environment by matching planned multiple views. In particular, we discuss the following aspects:

1. the decomposition of a framed view and the construction of partial 3-D descriptions of the view;
2. the matching of partial 3-D descriptions of a view with the built-in model of the robot environment;
3. the matching of partial descriptions of bodies derived from the current framed view with partial models constructed from previous views;
4. the identification of the new information in the current view and the updating of the models;
5. the identification of the unknown parts of the models which are being constructed so that further vantage viewpoints can be planned.

This approach combines such intelligent robot functions as attention, planning, sensing, learning and knowledge rectification. A prototype system for matching and constructing 3-D body models has been implemented and tested with synthesized images using C-PROLOG under Berkeley UNIX on a VAX 11/750.

INTRODUCTION

In the future computer-controlled robots will be called upon to play many important roles in industrial, business and domestic situations. If these robots are to work in complex environments it will be necessary to develop knowledge-based sensory systems. In simple situations, the robot vision system can have built-in models of both the environment and all objects within it; this allows a relatively simple recognition process. In

more realistic situations, however, although the geometry of the surrounding environment may be known (i.e. the dimensions of the room, warehouse, etc, in which the robot operates), the type and position of the objects in the environment will generally be unknown. Thus knowledge of the structure and placement of these objects must be learned. To do this the mobile robot must first construct 3-dimensional models for the objects it encounters. It should then be possible to classify these objects by comparing their structural properties with those of generally known classes of objects such as benches, chairs, tables, etc.

In analysing a single framed view of part of a large scene, the problems which will generally stand in the way of constructing the 3-D body models include:

1. partial features;
2. self-occlusion;
3. occlusion;
4. accidental alignment and special alignment;
5. undetermined geometric parameters.

An approach to understanding a scene from image sequences by incrementally constructing body models seems promising. However, even to-day, the information processing load involved in analysing a sequence of images presents a serious technical problem. Dynamic selection of a minimal set of vantage viewpoints and effective selection of only the necessary information will be essential if the burden of computation is to be lightened. Fortunately, a mobile robot, by its nature, offers a good foundation for gathering information from different points of view. Thus combining a vision system with a planner, so that a scene can be analysed from planned multiple views, is both natural and necessary.

In this paper, we describe a system which incrementally constructs 3-D object models of an office or warehouse scene from planned multiple views. In particular, we address the matching and construction of 3-D partial models.

To limit the scope of the immediate research problem the following assumptions have been made:

1. The bodies in the environment are static, rigid, weakly externally visible, and have vertices formed by at most three surfaces. Edges are formed by two surfaces, which can be planar, conical, cylindrical or spherical.

This work was supported in part by grants from the NSERC of Canada. Mr. Xie was supported by the C. D. Nelson Memorial Scholarship.

2. The only lights in the environment are one point source and one diffuse source.
3. The shape and dimensions of the robot environment are given.
4. A pinhole spherical camera model is used to acquire the images.
5. There is a preprocessor which deals with early and intermediate vision processing of the visual data. The output of the preprocessor is equivalent to a complete 2 1/2 D sketch. The categories of facets and lines, the orientations of planes and the rough depths of junctions have already been extracted from the 2 1/2 D sketch.

Partially constructed models which remain incomplete after a sequence of views are analysed by a viewpoint planning system, which is described in a companion paper¹; using this system new views can be chosen to resolve the ambiguities. In any realistic situation, we would expect that the task assigned to the robot would also provide input to the planning system so that a decision could be made to ignore incomplete objects which were irrelevant to the current task.

The body models (either partial or complete) which are constructed from the multiple views have Boundary Representation (BR)-like representations. Once a complete model has been constructed, a rule based conversion system which has been described elsewhere² is used to transform the BR representation into our new "Constructive Solid Geometry - Extended Enhanced Spherical Image" (CSG-EESI) representation which provides both structural and geometric information for the bodies. Higher level 3-D models can be more easily derived from the CSG-EESI representation. This facilitates object classification by comparing a structure with those for prototype objects which might be expected to be in the environment.

BACKGROUND

There has been considerable research on the segmentation and labeling of images. After Guzman³, Huffman⁴, Waltz⁵ and Turner⁶, Chakravarty⁷ generalized a line and junction labeling scheme that deals with planar-faced or curved-surface solid bodies, having vertices formed by three surfaces. In this scheme, 3 types of lines and 8 types of junctions were defined. By dealing with regions and lines, objects can be correctly labeled by the set of junction types.

An object must often be observed from several directions in order to form an assessment of what it looks like or to form a 3-D model of it. In order to form a 3-D model of an object from sequential views, Underwood and Coates⁸ developed a program which forms a 3-D description of a planar convex object when the object is rotated in space. In their match algorithm, the connections between surfaces, the number of edges which bound a surface and the clockwise ordering of edges form the deterministic factors. Later, Preiss⁹ described an approach which interprets a standard engineering drawing of a planar object for construction of its 3-D representation. This approach consisted of three main steps:

1. interpretation of projected faces,
 2. interpretation of dashed lines,
 3. assembling them into a body.
- The connectedness properties and the geometric relationships between junctions or faces (such as coplanar relationships), were used for matching junctions. In this approach, the final 3-D representation of a body is complete, and consists of each of its faces, edges and vertices along with the three coordinates of each vertex.

Several researchers have investigated the problem of matching multiple views of a block's world in front of a featureless background. Using wide-angle stereo, Ganapathy¹⁰ designed a scheme which uses some heuristic rules, such as "Single Match", "Order Match", "Connectivity" and "Table Match", to choose an initial match between corresponding vertices in order to reduce the search space. His program finally stops after building up the 3-D coordinates of the vertices. Shapira and Freeman^{11, 12} developed a program for constructing a description of solid bodies from a set of pictures taken from different vantage points. A heuristic procedure was devised for establishing matches between junctions in the different pictures and determining the validity of doubtful junctions. It first establishes matches among junctions by using the constraints of projection and the connectedness between junctions; then it establishes matches among lines by using the cyclic-order property and fills in missing connections between junctions and missing junctions. The final description reported by the program involves bodies made up of their face groups which are described in terms of triples of matched lines. Asada¹³ developed a system which describes 3-D motions of jointed trihedral blocks. In this system, a Huffman-like labeling scheme and an object-to-object matching method are first used to segment the line-drawing images into individual blocks and to find the possible correspondence of their junctions between closely consecutive frames. A transition table of junction labels and contextual information is used to analyse structural changes of the line drawings. Then, the shape rigidity property of three vertices on a block is used to evaluate geometrical parameters, such as the 3-D coordinates of the vertices and motion parameters.

It is only in recent years that attempts have been made to match multiple views in a complex environment in order to incrementally construct some kind of model of a scene. Herman, Kanade and Kuroe¹⁴ described the 3-D MOSAIC project whose goal is to incrementally acquire a 3-D model of an urban scene from images. Their method is to first extract 3-D shape information from the images by stereo analysis, then to match two views based on junction matching and finally to generate an approximate model of the scene by using task-specific knowledge. Crowley¹⁵ described a navigation system for an intelligent mobile robot which included techniques for the construction of a line segment description of a recent sensor scan and the integration of such descriptions to build up a model of the immediate environment using a list of directed line segments. Herman¹⁶ described an algorithm which matches vertices in two 3-D descriptions. The algorithm consists of three main steps: first, initial matches are

obtained for each vertex based on local properties; next, a Waltz-filtering procedure is applied which propagates topological constraints to reduce the set of matches; finally, a tree-type search which uses both topological and geometrical constraints gives globally consistent sets of unique matches.

OVERVIEW OF THE SYSTEM

The system which we have developed for incrementally constructing 3-D models of objects is illustrated schematically in Figure 1.

The incremental construction of object models from

planned multiple views involves the following principal elements:

1. the decomposition of a framed view and the construction of partial 3-D descriptions of the view;
2. the matching of partial 3-D descriptions of a view with the built-in model of the robot environment;
3. the matching of partial descriptions of bodies derived from the current framed view with those partial models constructed from the previous views;
4. the identification of the new information in the current view and the updating of the models;
5. the identification of the unknown parts of the models which are being constructed so that further vantage viewpoints can be planned;
6. the finding of the relationships between bodies and

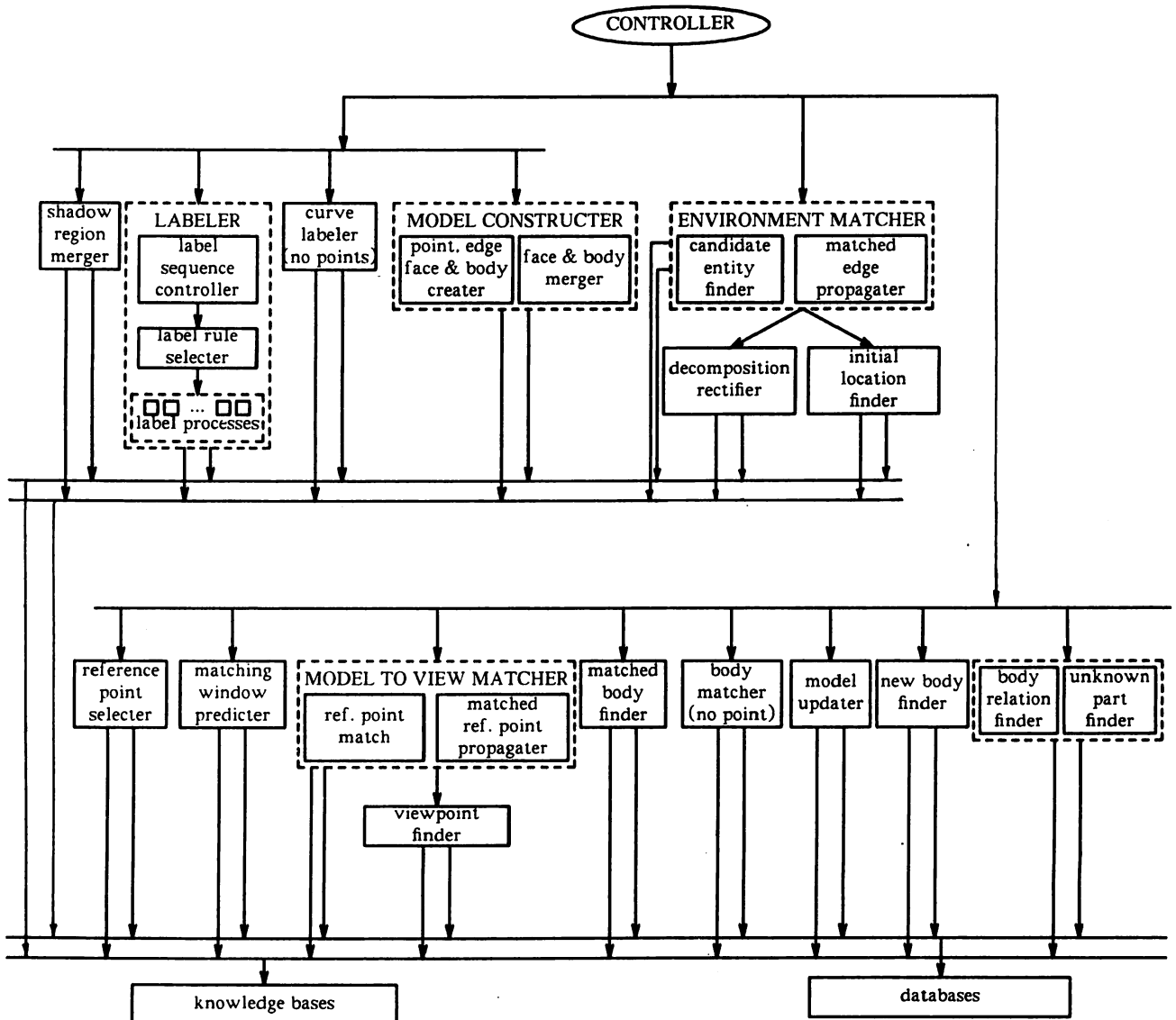


Figure 1

A schematic sketch of the system for matching and model construction.

their environment and constructing a partial map of the scene.

In the following sections we discuss elements 1 - 5 in turn. Each of these elements is more or less related to matching. In our approach, matching is based on the rules (constraints) derived from geometry, topology, photometrics, triangulation and problem assumptions.

Strategies for recognition can be data-directed (bottom-up), knowledge-directed (top-down) or some mixture of the two. In our approach, a bottom-up and top-down mixed strategy is used to match partial 3-D descriptions of a view with a built-in model of the robot environment and a data-directed strategy is used to incrementally construct 3-D body models for the objects in the environment. We are interested in exploring how far we can go with a data-directed strategy.

In the scene learning process the robot vision system generally produces incomplete and erroneous knowledge of the objects in the environment; consequently, it is important to identify the unknown parts, to rectify the erroneous knowledge and to assimilate the new information. Our approach combines such functions of an intelligent robot as attention, planning, sensing, learning and knowledge rectification.

DECOMPOSITION

For each view, the partial 3-D descriptions of bodies are derived by labeling and segmenting the image. In general, the decomposition process first merges the regions that are separated by shadow lines, and then it labels the junctions and lines in the image. After labeling, internal representations are created for real vertices, edges and faces. At that time, those edges separated by virtual junctions are united, and the partly viewed edges and faces are identified. In the last step, faces are combined on internal edges to form bodies. Some relationships between bodies, such as "touch by" or "occluded by" will also be identified. Thus, hierarchical internal representations which are used as partial 3-D descriptions are constructed for each body in a view.

Starting from the Chakravarty and Waltz labeling schemes, a modified and extended labeling scheme has been devised for labeling scenes containing shadows and certain curved objects. The images are labelled by an expert system; knowledge of label (production rules) is stored separately in micro-knowledge-bases according to the categories of related junctions. The top level of the expert system controls the sequence of the labels. It first arranges the addresses of the junctions that need to be labeled in an "ORDER QUEUE". The junctions which are generally easier to label, such as "p", "w" and "y" types, are arranged at the front of "ORDER QUEUE". When a junction and its related lines have been successfully labeled, the junction is deleted from the "ORDER QUEUE". Meanwhile, its immediately adjacent junctions will be inserted in a "PRIOR QUEUE". The top-level expert then propagates the labeled junctions to their immediate neighbours, thus the label procedure can take advantage of derived facts and the labeling time can be reduced.

The second level of the expert system selects the appropriate micro-knowledge-base, according to the category of the junction given by the top-level expert, and sequentially selects the production rules from the selected micro-knowledge-base in order to label the junction and its related lines. At the lowest level of the system are the processes which carry out the following tasks:

1. fetch the related facts from the appropriate micro-databases;
2. match the facts with the condition of a rule;
3. incorporate the label results into the appropriate micro-databases;
4. add the addresses of the immediate neighbours of the successfully labeled junction into the "PRIOR QUEUE".

The decomposition is conservative, i.e., it favours the separation of objects on concave edges. For a curve on which there is no feature point, a label is given according to its convexity: a concave curve is assigned as a concave boundary and a convex curve is assigned as a convex internal edge. The errors caused by an incorrect decomposition are expected to be rectified by facts collected later or by the knowledge stored at higher levels.

MATCHING THE ENVIRONMENT MODEL

The initial location of the robot in the environment is not known *a priori*. In order to determine the initial coordinates of the robot in a fixed coordinate system keyed to the environment, it is necessary to identify which entities in a framed view correspond to parts of the environment model. For this purpose, at least some edges of an entity should be matched with a connected part of the environment. Edges are stable, relative features which contain dimensional information and are at the lowest level (except for vertices). Since the geometry (shape and dimensions) of the environment are known, an edge-based matching process has been devised for matching the environment.

The process first matches the completely visible real edges of entities from a framed view with the built-in environment model; this is done according to their attributes, the categories (e.g. planar or conical) and the directions of their adjacent faces. The edge attributes consist of:

1. category, e. g., straight line, circle or other curve;
2. type, e.g., shadow, occluding boundary, concave internal, convex internal, concave boundary, clipping line or limb;
3. convexity;
4. approximate length.

If an edge in a view is matched with several edges in the environment model, then a "matching confidence" will be assigned to it which is proportional to the inverse of the number of matched pairs. An entity is considered to be a candidate for part of the environment model if at least its visible and well labelled internal and occluding edges match with edges in the environment model. From these candidates, entities will be designated as being parts of the environment on the basis of the following properties:

1. at least two matched edges;

2. a maximum number of matched edges;
3. a maximal sum of confidences for matched edges.

Following this identification, a top-down analysis process, which propagates the matched facts according to the built-in model of the environment, will be applied to those entities in order to:

1. Further verify the matched facts and find more matching facts. If in the propagation an inconsistent fact is discovered, then the initially matched entity will be rejected.
2. Identify the matched vertices and determine the position of the current viewpoint of the robot.
3. Rectify the result of the decomposition of the current view. When those concave edges, which were initially labeled as the concave boundaries, are revealed as the concave internal edges of the environment, their labels are revised and the corresponding bodies are merged together.

Since the 3-D coordinates of two known feature points and their spherical coordinates in a view can be used to determine the position of a viewpoint, the position can be determined from any two matched edges. From a pair of matched edges and the approximate depths of their related points (e.g. end points), the process identifies two pairs of the best matching points. Since the 3-D coordinates of the environment points are known, the possible position of the current viewpoint can be calculated from the two pairs of points. After another pair of matching edges has been discovered by the matching propagation procedure or when the other pair of matched edges is used for propagation, the new facts will be used to confirm or rectify the position of the viewpoint and make it more precise.

MATCHING PARTIAL BODY MODELS

Once the environment model has been matched, the partial 3-D descriptions from the first view will be used as the initially constructed partial models of the bodies in the scene.

In order to match the partial descriptions of bodies derived from a new view with those partial models constructed in the previous views, a multi-level feature matching approach has been used. This approach first matches the partially constructed models to those 3-D descriptions in the current view by selecting those reference vertices from the object models and the environment model which have the following features:

1. they are valid vertices or Shadow Intersection Points (SIP);
2. they are within the new view frame (though sometimes they may be occluded and unseen);
3. for each reference vertex, either the directions of the two constituent faces are known or the projection of the vertex is a boundary point of an unknown area in the current constructed map;
4. they are related to the objects of current interest.

Following the selection of the reference vertices a prediction process determines the possible matching windows in the new view; this is done on the basis of the approximate position of the new viewpoint and the coordinates of the reference vertices which may only

have approximate values stored in the models. The process finds the candidates for matching in the new view which are located within the matching windows. The widow sizes are determined by the tolerance errors of the robot movement, the errors of the coordinates of the reference points and the relative positions of the new viewpoint and the reference points.

In the knowledge base, there is a "Junction Family Dictionary". In the dictionary, each family consists of the possible junction types for a specific kind of vertex, when it is viewed from different positions. Using the Junction Family Dictionary, and the categories and directions of the constituent faces, the matching process assigns each candidate a confidence. The candidate which has the uniquely highest confidence will be chosen as the matched vertex for a reference vertex, and its corresponding faces will be considered as matched faces. After finding a matched pair of vertices, the matching process propagates the fact along the emanating edges to adjacent vertices. A depth-first search is used at each pair of matched vertices, and when partial edges, unmatched vertices caused by occlusion or already matched facts appear, the match propagation for that direction will stop. Thus, different levels of features (i.e. faces, edges and vertices) can be matched in the same propagation process.

For those constructed body models which do not have any vertex or feature point (e.g. SIP), the edges and faces will be chosen as the basic matching elements. According to their categories, types, shape parameters and approximate positions, the corresponding feature elements can be found. Also the matched facts will be propagated to their related feature elements.

After the faces have been matched, matched bodies can be derived from these faces. Following this, the model updating process searches the matched facts starting with high level features and moving to low level features; the low level features which do not exist in the body model are now filled in by the known parts of the current 3-D body description which are matched. After matching the partially constructed models to those 3-D descriptions in a new view, unmatched bodies in the new view are identified. In order to test whether these are newly discovered bodies, a reverse direction matching process is used to check whether any vertex in an unmatched body has a corresponding vertex in the body models or the environment model. If this is not the case, then the body is new and is added to the database of the body models; otherwise the appropriate matched model will be found. Meanwhile, features separated in the new view or in the constructed models may be merged into one if their correspondence is unique in one of the two 3-D representations. The related revision will also be done.

POSITION ADJUSTMENT

In practice, data gathered by a robot vision system always includes certain tolerance errors. Although the relative positions of the viewpoints can be derived from a robot servo system, this information is generally imprecise. Since any two views form a pair of wide angle stereo images, the matching process provides the

information necessary to calculate the position of the sensor (the robot camera) quite precisely. This information can be used to correct the position calculated by the movement control servos and used for dynamically adjusting the robot movement.

IDENTIFICATION OF UNKNOWN PARTS

The ambiguities caused by special alignments and accidental alignments generally can be distinguished by using multiple views. For example, when a strange junction type occurs in a view, if from other views the matched points belong to the same family of junctions, then it is caused by a special alignment, otherwise it is caused by accidental alignment. The ambiguity caused by accidental alignments can be ignored. For a special alignment, the ambiguity can often be solved by a correct decomposition, though sometimes higher knowledge of the scene may be needed.

Inside a body, self-occlusion may result in unknown occluded parts. Between bodies, an occlusion may cause the occluded bodies to be unidentified. For these two cases, unknown parts only occur at the occluding edges. Besides, a concave boundary edge indicates that the two related bodies are touching, and hence the touching parts cannot be seen if there is no means to change the status of the bodies.

In the system described here, when a model of a body has been created, only the internal, occluding and concave boundary edges which are the real edges of the body are created. The model also contains a list of its boundary edges and a list of the bodies which occlude it. When a newly discovered surface is added into a body model, it is necessary to change the types of those occluding edges in the body model, which are matched with the edges of the added surface. These edges become the internal edges of the body and are deleted from the boundary list of the partial model of the body. Thus boundary occluding edges of a body model always indicate the self-occlusion of parts and the need for further attention.

The "t" type junctions caused by occlusion are kept in the input image databases. Although they are not the vertices of a body, they are important points for the construction of a map of the scene and for discovering the unknown parts caused by occlusion. For an occluded body, discovering its occluded parts is accompanied by a search for its "t" type points and those incompletely seen edges and surfaces which relate to the "t" type points.

All of the above outcomes will be organized and analysed by a view planning system in order to further resolve the ambiguities. This component has not yet been developed and implemented.

EXPERIENCE

The system for matching and constructing 3-D body models has been implemented by using C-PROLOG under the UNIX operating system on a VAX 11/750.

Figure 2 shows two synthesized views from the scene shown in Figure 3; they have been successfully analyzed by the system described above.

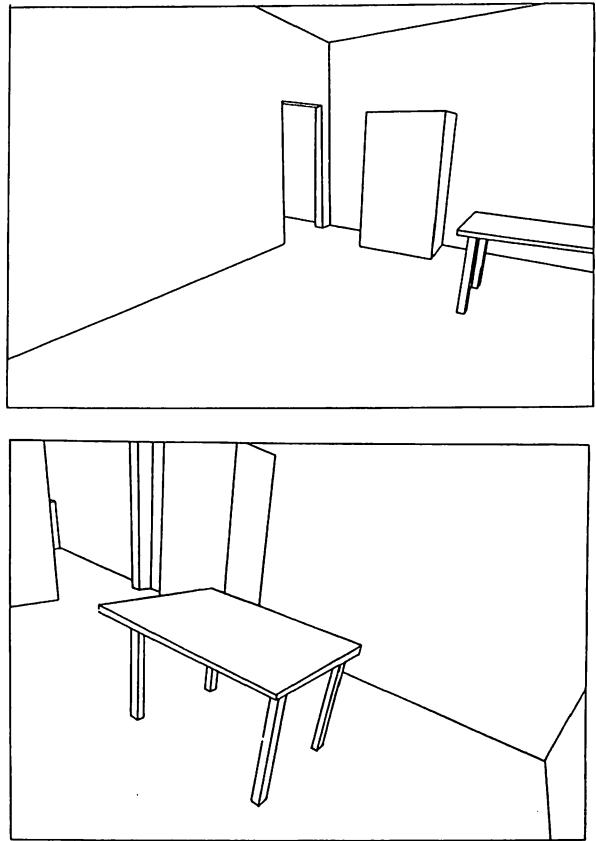


Figure 2

The two synthesized views which have been successfully analyzed by the described system.

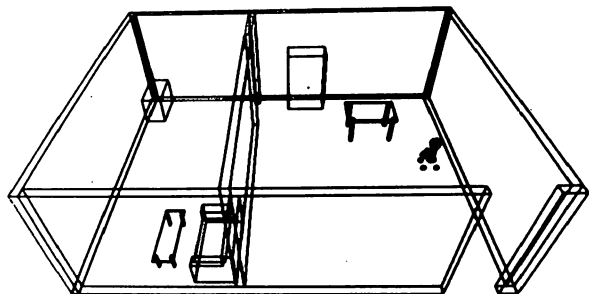


Figure 3

An example of a typical scene (two views of this scene are used in Figure 2).

CONCLUSION

Under the assumptions described in the introduction, the system described here can incrementally construct 3-D body models in an office or warehouse environment by matching planned multiple views. No prior knowledge of the objects is required by this system. The system includes the following important features:

1. a framed view is decomposed and partial 3-D descriptions of the view are constructed;
2. partial 3-D descriptions of a view are matched with the built-in model of the robot environment;
3. partial descriptions of bodies derived from the current framed view are matched with those partial models constructed from the previous views;
4. the new information in the current view is identified and the models are updated;
5. the unknown parts of the models which are being constructed are identified so that further vantage viewpoints can be planned.

Together with a view planning system¹ and the CSG-EESI 3-D model conversion system², this system offers a good basis for constructing a higher level image understanding system for an intelligent robot.

As noted above, the system has been implemented in C-PROLOG under the UNIX operating system on a VAX 11/750, and has been tested successfully with synthesized images. While C-PROLOG provides a good environment for testing the ideas used in this system, any practical implementation would have to be more efficient.

REFERENCES

1. S. Xie, T. W. Calvert and B. K. Bhattacharya, "Planning views for the incremental construction of body models", *submitted to The 8th International Conference on Pattern Recognition*, Paris, France, 1986.
2. S. Xie and T. W. Calvert, "The CSG-EESI scheme for representing solids with a conversion expert system", *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, San Francisco, Ca., 1985, pp. 124-129.
3. A. Guzman, "Decomposition of a visual scene into three-dimensional bodies", *Proc. AFIPS Fall Joint Compt. Conf.*, vol. 33, 1968, pp. 291-304.
4. D. A. Huffman, *Impossible objects as nonsense sentence*, Edinburgh Univ. Press, Edinburgh, U.K., 1971.
5. D. L. Waltz, *Understanding line drawings of scenes with shadows*, McGraw-Hill, New York, 1975, pp. 19-91.
6. K. J. Turner, "Computer perception of curved objects", *presented at the AISB Summer Conf.*, Univ. Sussex, Brighton, England, 1975.
7. I. Chakravarty, "A generalized line and junction labeling scheme with applications to scene analysis", *IEEE Trans. on PAMI*, Vol. PAMI-1, 1979, pp. 202-205.
8. S. A. Underwood and C. L. Coates, "Visual learning from multiple views", *IEEE Trans. on Computers*, Vol. C-24, No. 6, June 1975, pp. 651-661.
9. K. Preiss, "Algorithms for automatic conversion of a 3-view drawing of a plane-faced part to the 3-D representation", *Computers in Industry*, Vol. 2, 1981, pp. 133-139.
10. S. Ganapathy, *Reconstruction of scenes containing polyhedra from stereo pair of views*, PhD dissertation, Stanford Artificial Intelligence Laboratory, Memo AIM-272, December 1975.
11. R. Shapira, "Reconstruction of curved-surface bodies from a set of imperfect projections", *Proc. of the 5th IJCAI*, 1977, pp. 628-634.
12. R. Shapira and H. Freeman, "Computer description of bodies bounded by quadric surfaces from a set of imperfect projections", *IEEE Trans. on Computers*, Vol. C-27, No. 9, Sept. 1978, pp. 841-854.
13. M. Asada, *Understanding of three-dimensional motions in trihedral world*, PhD dissertation, Dept. of Control Engineering, Osaka University, Japan, Jan. 1982.
14. M. Herman, T. Kanade, and S. Kuroe, "Incremental Acquisition of a Three-Dimensional scene model from images", *IEEE Trans. on PAMI*, Vol. PAMI-6(3), May 1984, pp. 331-340.
15. J. L. Crowley, "Dynamic world modeling for an intelligent mobile robot using a rotating ultrasonic ranging device", *Proc. IEEE Inter. Conf. Robotics and Automation*, St. Louis, Miss., 1985, pp. 128-137.
16. M. Herman, "Matching three-dimensional symbolic description obtained from multiple views of a scene", *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, San Francisco, Ca., 1985, pp. 585-590.

IMAGE TECHNIQUES FOR THE IDENTIFICATION OF DEPRESSIONS AND
OTHER OBSTACLES IN AUTOMATED GUIDANCE OF ROVING ROBOTS

Malek Adjouadi
Department of Electrical Engineering
University of Hawaii at Manoa
Honolulu, Hawaii 96822

ABSTRACT

In this study, we analyze the problem of detection of depressions or drop-offs in the automated guidance of roving robots. The proposed approach is based on the principle that if one is too near a depression, one is bound to see new information which initially was occluded. To exploit this principle, two steps are undertaken. The first step involves the derivation of the correspondence process to allow the vision system to relate a location of interest in a sequence of frames. The second step involves the development of methods to detect and identify, in this location of interest, the occluded information.

RÉSUMÉ

Nous présentons dans cette étude une méthode visuelle pour la détection de dépression de terrain (creux, pente, bordure de pavé, etc) dans le but d'assurer sauf-conduit pour un robot autonome. Cette méthode est basée sur le principe que si on s'approche d'une dépression de terrain, on est apt d'apercevoir certains éléments d'information qui auparavant n'étaient pas apparents. Pour exploiter ce principe, deux étapes d'action sont prises. Dans la première étape, nous établissons le rapport qui existe entre une série de photographes pour permettre au robot de reconnaître une même location d'intérêt dans deux ou plusieurs de ces photographes. Ces photographes sont prises en succession et en s'approchant de cette location d'intérêt. Dans la deuxième étape, nous décrivons les techniques nécessaires pour identifier et extraire les éléments d'information qui caractérisent la présence d'une dépression de terrain.

INTRODUCTION

Depressions or drop-offs constitute a serious problem in the automated guidance of roving robots. Unfortunately, the detection of depressions is also a complex image analysis problem. In the human vision system, many visual cues such as stereopsis, occlusion cues, context in the viewed scene, change in textural properties, etc., are all interpreted and integrated with relative ease to yield an almost effortless perception of what, in fact, is a complex perceptual task. In image processing, however, a computer implementation exploiting any one of the aforementioned cues becomes a complex information processing problem.

Clearly, there is no simple way to solve this problem. In the approach proposed here, the aim is to extract the occluded information given a sequence of frames based on methods which allow for a relaxed image correspondence process between these frames. The two methods devised here make use of intensity profiles or pixel intensity distributions. The first method identifies primary cues which suggest the presence of a depression. The second method extracts the occluded information to confirm the presence of a depression.

Before we describe the approach for the detection of depressions, we first take a broad view of the automated guidance of roving robots. In this view, with the aim to extend beyond the ideal settings generally considered, we make an assessment of real-world scenes identifying pertinent problems towards enhanced guidance of roving robots.

SCENE INTERPRETATION PROCESS

Figure 1 illustrates a process of analysis and interpretation of real-world scenes. In this process, the first function of the vision system is to provide the robot with a safety path. To carry out this function, the vision system uses the first-pass evaluation process¹ which exploits the surface consistency constraint² by comparing the environment ahead of the robot with an initial environment which is already determined to be obstacle-free. Computer results of an implementation of this first-pass evaluation on outdoor scenes are shown in Figure 2. The second function of the vision system is to provide the robot with the needed additional information in the event where an object blocking the path of travel is detected, or if some landmark need to be identified. To carry out this second function, the objects the robot is likely to encounter are categorized, and their essential visual characteristics are identified. In the process of Figure 1, these categories are: (1) shadows (false alarm), (2) depressions, (3) upright objects, and (4) flat objects. The essential features characterizing the above categories are:

1. Shadow. A surface upon which a shadow is cast will preserve its intrinsic physical characteristics. This is due to the relatively uniform effect of shadow on the image gray level intensities.^{3,4}

2. Depressions. In approaching a depression, one is bound to see new information that was previously occluded. We call this the occluded information.
3. Upright objects. Most man-made upright objects have straight vertical edges. The few other man-made objects together with the natural objects which do not have straight vertical edges can be distinguished by the manner in which they project onto the two-dimensional (2-D) image plane. An upright object projects in the 2-D image plane proportionally to the depth of field it occludes.⁵
4. Flat objects. Flat objects are affected by perspective. Also, a flat object projects in the 2-D image plane proportionally to its length.⁵

A methodology for guiding the robot through a given scene can be:

1. As an initial step, the vision system takes left, front and right images of the scene to acquire a wide-angle view. Each image is analyzed using the first-pass evaluation process. The results obtained from the three images are integrated to yield an optimal tracing of the safety path. We refer to this step as the initialization phase.
2. The robot takes the optimal path, and the vision system is directed to enter what we refer to as the motion phase. In this motion phase, the vision system processes images in the chosen direction of travel. The wide-angle view is no longer necessary, unless an obstruction is encountered and a new direction of travel must be taken. Moreover, the image taking process is a function of the range of the safety path. For example, if a path is obstacle-free for x steps, then an image may be taken every x_1 steps, where x_1 is the integer part of the fraction x/k and $k=2,3,4,\dots$ depending on how large x is. In this phase, essential safety path cues such as path clear, obstacle ahead, turn left/right, are provided in real-time, and the timing of the vision system is such that it is always processing a few steps ahead of the robot.
3. If an object is found along the direction of travel, the vision system issues a warning signal to the robot and directs it to pause and takes another picture. The system then determines the range and extent of the object, and provides the robot with the necessary avoidance cues. If identification of the object is desired, the system enters the identification process. We refer to this step as the warning/identification phase. This phase can be carried either in a sequential mode or in a parallel mode. In the sequential mode, the processing task for which the primary cues can be found with the least amount of processing time is performed first. If the results are not conclusive, the next processing task is performed, and so on. In the parallel mode, all processing tasks are initiated simultaneously, and execution of these tasks ends as the first primary cues are determined.

In organizing all these information processing tasks to yield an integrated vision system, the following important points are considered:

1. Implementation of a methodology and a decision making process to insure that an information processing task is initiated only if primary cues justify its execution;
2. Allowing for concurrent processing in the development of these information processing tasks;
3. Allowing for one processing task to call upon another processing task if ambiguities arise in the image interpretation results.

We now describe the approach for the detection of depressions. This description starts with a presentation of the image correspondence process.

IMAGE CORRESPONDENCE PROCESS

For the image correspondence process, we derive equations for the correspondence, in both range and width, of any two image points, going from one frame to the next. But first, we need to define the mapping principles between the three-dimensional (3-D) real world and the 2-D image. Given Figure 3, using properties of similar triangles, measurements in width (W) and range (R) in the real-world environment are mapped in the (x,y) image coordinate system by the following relationships:

$$y_k = \frac{f[R(y_k) + h \tan \alpha] + f[R(y_k) \tan \alpha - h] \tan(\beta + \alpha)}{[R(y_k) + h \tan \alpha] \tan(\beta + \alpha)} \quad (1)$$

$$x_j - x_i = \frac{f + [R(y_i) - R(y_0)]}{fW(x_i, x_j)} \quad (2)$$

where h is the camera height; f is the camera focal length; α is the camera tilt angle; and $\beta = \arctan(L/h)$, with L being the range between the camera and the first point viewed by the camera. Note that if we let $\alpha = 0$, Eq. (1) takes the simple form

$$y_k = \frac{fh[R(y_k) - L]}{LR(y_k)} \quad (3)$$

1. Range correspondence: The objective here is to find how two points, y_{l1}^1 and y_{l2}^1 , in the vertical axis of the first frame map into points, y_{l1}^2 and y_{l2}^2 , in the second frame. The difference in the range of the two frames is r_1 . The superscript, 1 or 2 denotes the frame identity. Coordinate y_{l1}^1 is the point where the object area starts, and point y_{l2}^1 is an arbitrary point a small distance away from y_{l1}^1 . The distance between these two points depends on the extent of the detected object. For simplicity, let us assume that the tilt angle α of the camera plane is zero. To find the mapping between points (y_{l1}^1, y_{l2}^1) and (y_{l1}^2, y_{l2}^2) , we use Eq. (3) to obtain the following relationships:

$$y_{l1}^1 = \frac{fh[R(y_{l1}^1) - L]}{LR(y_{l1}^1)} \quad (4)$$

$$y_{l1}^2 = \frac{fh[R(y_{l1}^1) - r_1 - L]}{L[R(y_{l1}^1) - r_1]} \quad (5)$$

where $R(y_{l1}^1)$ is estimated by the first-pass evaluation. Similarly, since y_{l2}^1 is arbitrarily chosen, its range $R(y_{l2}^1)$ is easily determined, and thus we obtain

$$y_{l2}^1 = \frac{fh[R(y_{l2}^1) - L]}{LR(y_{l2}^1)} \quad (6)$$

$$y_{l2}^2 = \frac{fh[R(y_{l2}^1) - r_1 - L]}{L[R(y_{l2}^1) - r_1]} \quad (7)$$

2. **Width correspondence:** In order to save processing time, it is useful to focus only on a certain width of the image where the object is detected. So, if we choose a segment of width delimited by x_{k1} and x_{k2} in the first frame, we need to find their corresponding projections in the second frame. This correspondence is found using Eq. (2). By substituting $R(y_{l1})$ for $f + [R(y_1) - R(y_0)]$, for the first frame, we have

$$x_{k1}^1 - x_{k2}^1 = \frac{fw(x_{k1}^1, x_{k2}^1)}{R(y_{l1}^1)} \quad (8)$$

For the second frame, we have

$$x_{k1}^2 - x_{k2}^2 = \frac{fw(x_{k1}^1, x_{k2}^1)}{R(y_{l1}^1) - r_1} \quad (9)$$

Using these relations, the vision system can now relate a location of interest in any two distinct frames separated by an arbitrary range r_1 (see Figure 4).

EXTRACTION OF THE OCCLUDED INFORMATION

Occluded information that is revealed in a subsequent frame can be perceptually very deceptive (see Figure 5). This results from the fact that if we cannot locate the same point of reference in the two frames then we may conclude that there is no relationship between these two frames. This stresses the importance of a reference point from which the system starts to look for occluded information; in our analysis this reference point is chosen in the proximity of the object as indicated by the first pass evaluation. Moreover, the detection of occluded information will disturb many of the physical relationships that previously existed between the various elements in the given scene. Utilizing these two ideas, we describe two simple methods to extract this occluded information.

Method 1

- Step 1: Take a vertical scan from point y_{l1}^1 to point y_{l2}^1 to generate a vertical pixel intensity distribution or profile. We call this profile P_{l1l2}^1 . Similarly, generate a vertical pixel intensity profile, P_{l1l2}^2 , between y_{l1}^2 and y_{l2}^2 .
- Step 2: Locate the number of major disturbances (peaks) in the profile P_{l1l2}^2 when compared with the peaks in P_{l1l2}^1 . If there exists a difference in the number of major disturbances in the two profiles, then this implies the detection of the occluded information. By major disturbance or

peak, we mean a point in the profile whose gray level value exceed the value P_{max} given by the standard relation⁶

$$P_{max} = \mu_p + 0.5 \sigma_p$$

for the same profile. Parameters μ_p and σ_p are the mean and standard deviation of the intensity profile, respectively.

This method is used by the integrated vision system for determining the primary cues.

Method 2

- Step 1: Obtain a few horizontal scans between points x_{k1}^1 and x_{k2}^1 of the first frame, starting at the vertical coordinate y_{l1}^1 . We call these intensity profiles $P_{klk2}^1(i)$, where i denotes the i -th scan. Obtain similar scans from the second frame, using the coordinate y_{l1}^2 as the starting point and x_{k1}^2 and x_{k2}^2 as the horizontal limits. We call these intensity profiles $P_{klk2}^2(i)$.
- Step 2: As in step 2 of the previous method, locate a difference in the number of major peaks appearing in $P_{klk2}^2(i)$ when compared to $P_{klk2}^1(i)$. If peaks are found, occluded information is detected.

This method confirms the results obtained using the previous method.

Computer examples of this procedure are illustrated in Figure 6. Note, that when no occluded information is found in this analysis, the object remains a potential obstacle.

CONCLUSION

We described in this study an approach for the detection of depressions. This approach is based on finding occluded information from a sequence of frames. We noted that if this occluded information was not found, the object in question remains a potential obstacle, and the appropriate processing task is initiated to identify its nature. An attractive feature of this approach is that the methods used for the extraction of occluded information allow for a relaxed image correspondence process. For example, if a given peak in the intensity profile on an initial frame is missing in the corresponding intensity profile of a subsequent frame, it becomes sufficient to look for a disturbance in these peaks going from one frame to the next. A computer implementation of this approach on real-world scenes produced very good results. Moreover, in the first part of this study, we discussed an image interpretation process which identifies pertinent problems towards enhanced guidance of roving robots.

REFERENCES

1. J.T. Tou and M. Adjouadi, "Computer Vision for Roving Robots," Proceedings of the Cannes Symposium, Cannes, France, 1985.
2. W.E.L. Grimson, "A Computational Theory of Visual Surface Interpolation," Philosophical Transactions of the Royal Society of London, Vol. B298, pp. 395-427, 1982.

3. H.G. Borrow and J.M. Tenenbaum, "Recovering Intrinsic Scene Characteristics from Images," in Computer Vision Systems, A.R. Hanson and E.M. Riseman, Eds., Academic Press, New York, 1978.
4. J.T. Tou and M. Adjouadi, "Shadow Analysis in Scene Interpretation," Proceedings of the 4th Scandinavian Conference on Image Analysis, Trondheim, Norway, June 1985.
5. M. Adjouadi, "Discrimination of Upright Objects from Flat-Lying Objects in Automatic Guidance of Roving Robots," SPIE's Symposium Southeast on Optics and Optoelectronic Systems, Orlando, Florida, March 1986.
6. D. Brzakovic, "Computer Based 3-D Scene Description from Texture," Ph.D. Dissertation, Department of Electrical Engineering, University of Florida, 1984.

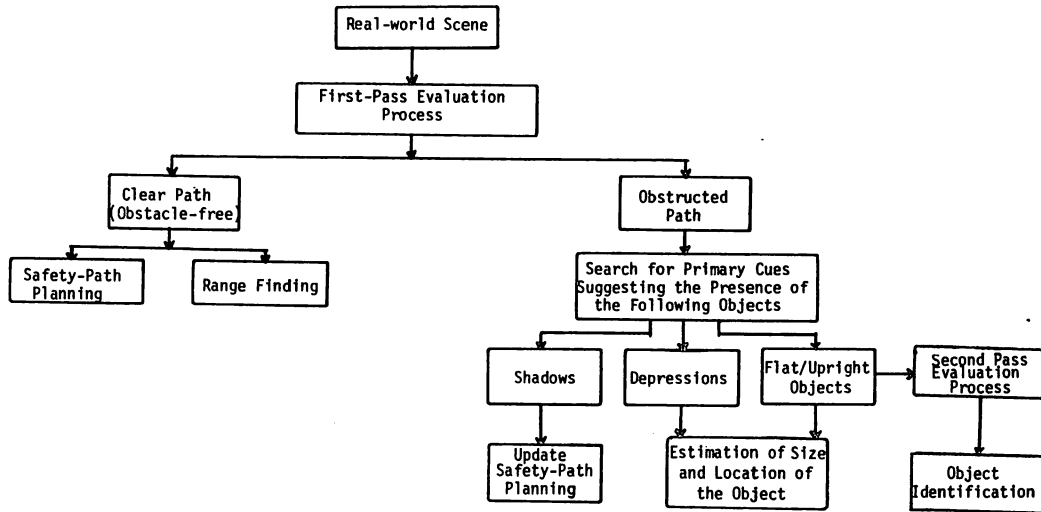


Figure 1. Process of Analysis of Real-World Scenes

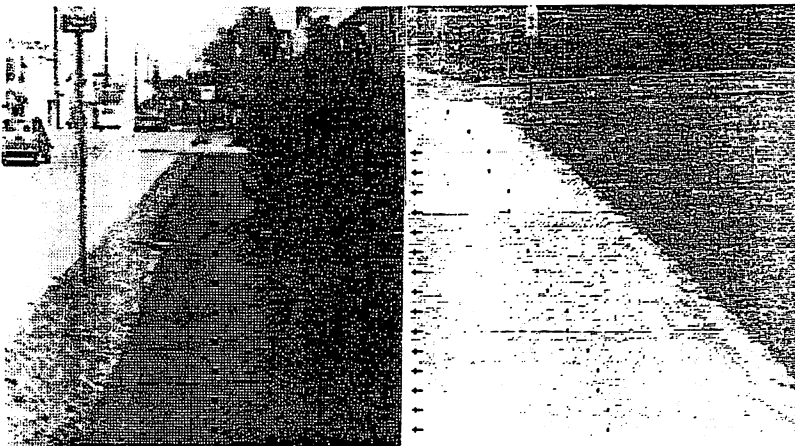
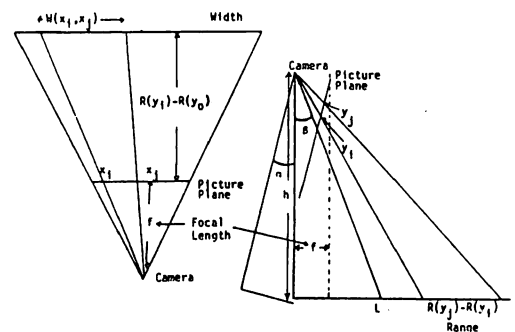


Figure 2. Results of the First-Pass Evaluation on Two Outdoor Scenes



(a) Mapping of Width

(b) Mapping of Range

Figure 3. Mapping of Real-World Measurements onto the Image Plane

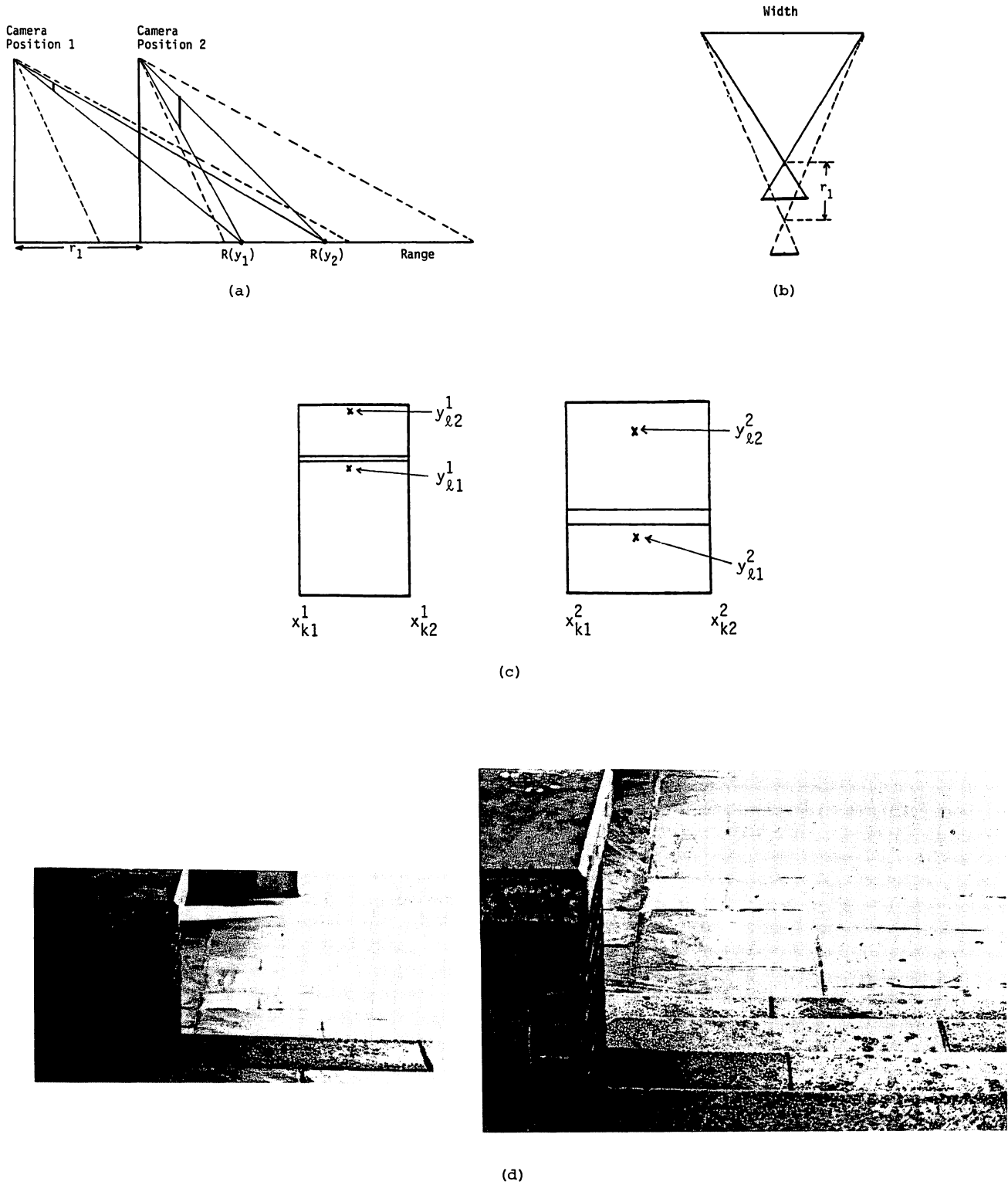


Figure 4. Image Correspondence for Extracting Occluded Information. (a) Correspondence in Range; (b) Correspondence in Width; (c) Mapping the Reference Points; (d) Mapping the Location of Interest.

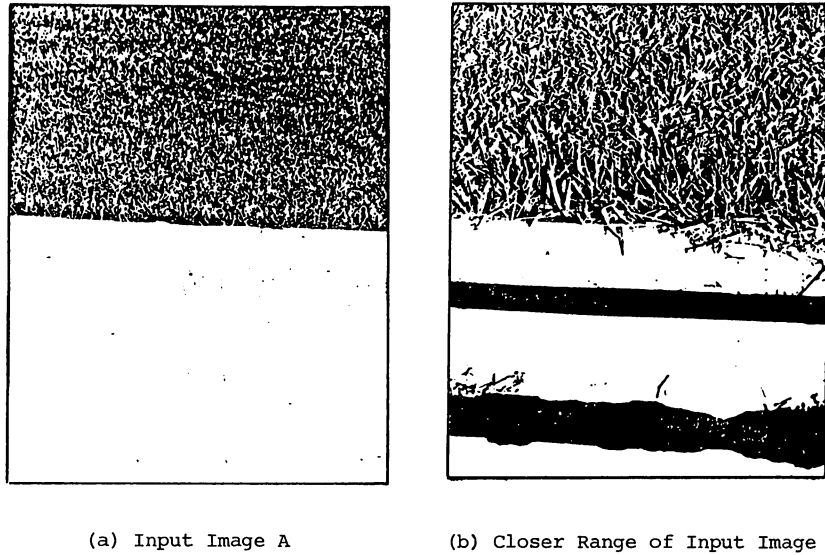


Figure 5. Need for a Point of Reference to Extract Occluded Information

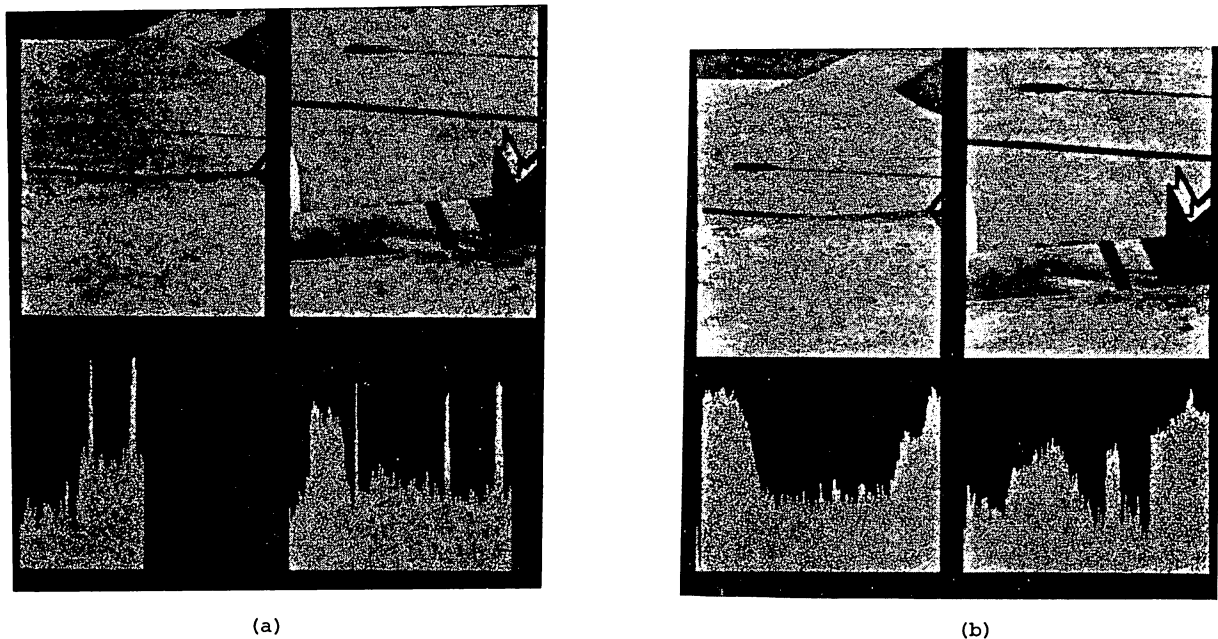


Figure 6. Extraction of Occluded Information. Input Images and their
(a) corresponding vertical intensity profiles with occluded information revealed in the close range frame;
(b) corresponding horizontal intensity profiles with occluded information revealed in the close range frame.

A COMPUTATIONAL THEORY OF 3D SHAPE RECONSTRUCTION

FROM IMAGE CONTOURS

Ping Liang and John S. Todhunter

Dept. of Electrical Engineering

University of Pittsburgh, Pittsburgh, PA 15261, U.S.A.

ABSTRACT

The computational theory of 3D shape reconstruction from image contours proposed in this paper is based on the variational principles and has the theoretical framework suggested recently by Poggio [8]. By this theory, 3D shapes can be reconstructed from image contours by general physical constraint assumptions, namely the assumptions of minimum potential energy, isotropism and homogeneity of the material, and properly defined energy functionals. It is assumed that contours have been classified as surface discontinuity boundary contours, surface contours and extremal boundaries. Minimization of the energy functionals tends to maximize the symmetry and orthogonalize the surface junctions of the reconstructed object. Some early findings are obtained as the natural results of the theory. Theoretical developments and experimental results are successful and promising.

EXTENDED SUMMARY OF THE THEORY

One important function of early vision is the reconstruction of a 3D representation of a scene from 2D images. Stereopsis and structure from motion are the most explored in vision studies. Stereopsis and structure from motion require multiple images, but humans have the ability to perceive (with illusion) the 3D environment with only one eye or from a single picture.

There exist many sources of information about surfaces in an image such as texture, shading, shadow, etc. [9,12,13,17], but those methods are only applicable for certain special situations. It has been shown that shape reconstruction from contours is significantly more powerful than shape reconstruction from textures [7]. Barrow and Tenenbaum [11] argued that shape reconstruction from boundary contours is of fundamental importance in explaining surface perception and more important than shape reconstruction from shading. Steven [14] showed that surface contours also play an important role in shape reconstruction from image.

Theoretical studies [8] showed that the computational, ill-posed nature of early vision problems leads naturally to the application of the mathematical theory of regularizing ill-posed problems for solving them in terms of variational principles that enforce general physical constraints derived from a physical analysis of the problem. The constraints

should be derived as a natural consequence of the physical laws governing the world we live in. The results of this research will not only impact our understanding of the early visual system in biological organisms, but also lead to development of computational algorithms and hardware designs for machine vision.

The fact that the human visual system has definite and consistent interpretations of contour images shows that it exploits some implicit assumption about the world. Without any knowledge of the nature of the process which generated the 3D shape, it is reasonable to assume that the given 2D contour is most likely to correspond to the projectively equivalent 3D shape with minimum potential energy. It is well known from classical mechanics that a physical system is stable if and only if its total potential energy is minimal. In many cases, it is also justified on the grounds that the surfaces tend to assume smooth and minimal energy configurations. Because there is no information available in the image contours about the material of the surface, the only reasonable assumption is that the surface material is isotropic and homogeneous. The computational theory of shape reconstruction from contours proposed in this paper is based on the variational principles in terms of general physical constraint assumptions: (1). the minimum energy principle; (2). isotropism and homogeneity of the material, i.e., the uniformity of the energy distribution.

To deform a system with minimum total potential energy to a non-minimum energy system, external energy has to be converted into potential energy in the system. As is well known, the differential equations describing a system in non-minimum energy state are far more complicated than the equations describing the system in its minimum energy state. Non-uniformity of energy distribution represents information about the system. Thus we can draw a correspondence between energy, energy distribution and information. Therefore the interpretation of the image contours by this theory is a minimum information interpretation in some sense.

Earlier studies such as [3,4,5,6,9,10,11,12,13,14,17] motivate and support the theory proposed in this paper. Barrow and Tenenbaum [11] optimized a smoothness measure to reconstruct planar curves and polyhedra. The optimization criterion used in [11] for continuous curves and straight lines are different, whereas a complete theory of shape

reconstruction from contours should be able to accommodate both cases. The optimization criterion developed in [1] by the authors, which is a preliminary version of the theory proposed in this paper, uses a single underlying mechanism for both continuous curves and straight line contours. Witkin [12] developed a maximum likelihood approach to shape reconstruction from contours, and achieved some success in interpreting irregular shaped objects. This method is ineffective when the contour has a regular shape and does not compute the right slant of an ellipse. Brady and Yuille [7] developed an extremum principle which maximizes the ratio of the area to the square of the perimeter. Their method would be ineffective to curved surface and images with both boundary and surface contours. The theoretical framework proposed by Poggio [8] lends strong support to the theory proposed in this paper.

Kanade [9,17] developed a systematic method to recover 3D shapes from a single view by mapping image geometric properties into shape constraints. He proposed the assumption of mapping 2D skewed symmetry into 3D symmetry, and proved that the skewed symmetry can be a projection of real symmetry if and only if its surface gradient is on a certain hyperbola in the gradient space. We have proved that Kanade's assumption and hyperbola are natural results of the theory we proposed. Barnard [10] recently proposed a maximal orthogonal principle for 3D recovery based on psychophysical data. This principle is further developed and incorporated into our theory.

Some work has been done which offers proof to the minimum energy principle approach of shape reconstruction. Grimson [5,6] and Terzopoulos [3,4] used a thin plate model and constructed the 3D surfaces from the scattered stereo depth data by minimizing the total potential energy of the thin plate. The work by Barrow and Tenenbaum [11] is based on a similar idea in interpreting line drawings by optimizing a "smoothness" measure.

The outline of the theory is summarized as follows. It is assumed that the contours and junctions have been classified as surface discontinuity boundaries and junctions, surface contours, and extremal boundaries. The classification itself is a very important problem, and there is still no complete solution to it. Orthographical projection is assumed throughout.

First the reconstruction of a single surface from a simple closed 2D boundary contour is considered. Suppose that the 2D boundary contour has continuous curvature $\kappa_0(t)$ and is given by $r_0(t) = (x_0(t), y_0(t))$, $t \in T = [a, b]$, $x_0(a) = x_0(b)$, $y_0(a) = y_0(b)$, where t is a parameter invariant under magnification or contraction. Backprojecting $r_0(t)$ into 3D as $r(t) = (x_0(t), y_0(t), z(t))$ such that $r(t)$ has continuous curvature $\kappa(t)$ and torsion $\tau(t)$. Let

$$\left(\kappa(t) \frac{ds}{dt}, \tau(t) \frac{ds}{dt} \right) = (\kappa_t(t), \tau_t(t)). \quad (1)$$

Define a vector

$$p(t) = \begin{cases} (\kappa_t(t), \tau_t(t); \kappa'_t(t), \tau'_t(t)), & \text{where } r_0(t) \text{ is convex.} \\ (\pi - \kappa_t(t), \pi - \tau_t(t); \kappa'_t(t), \tau'_t(t)), & \text{otherwise.} \end{cases} \quad (2)$$

Suppose the derivatives $\kappa'_t, \tau'_t \in L^2$. Let $A = \{r(t) = (x_0(t), y_0(t), z(t)), \kappa(t), \tau(t) \text{ continuous}, \kappa'_t, \tau'_t \in L^2\}$, $B = \{p(t) | r(t) \in A\}$. $C_c^0(\Omega)$ denotes the set of all continuous functions with compact support. Let H^m be the Sobolev spaces, and H_0^m be the completion of $C_c^\infty(\Omega)$ in the norm $\|\cdot\|_{m,\Omega}$. Note that for $p(t) \in B$, $\kappa_t(t), \tau_t(t) \in H^1$.

Define an inner product as

$$(p_1, p_2) = \frac{k}{2} \int_T \kappa_{t1}(t) \kappa_{t2}(t) dt + \frac{k}{2} \int_T \tau_{t1}(t) \tau_{t2}(t) dt + \frac{k}{2} \int_T \kappa'_{t1}(t) \kappa'_{t2}(t) dt + \frac{k}{2} \int_T \tau'_{t1}(t) \tau'_{t2}(t) dt \quad (3)$$

where k_κ, k_τ are called energy factors of curvature and torsion, and $k_{\kappa 1}, k_{\tau 1}$ are called uniformity factors of curvature and torsion. The integrals are Lebesgue's integrals. When $p_1 = p_2$, the first two terms are measures of the potential energy in the reconstructed shape, and the last two terms are measures of the uniformity of the potential energy distribution.

The assumption that $\kappa_t(t), \tau_t(t) \in H^1$ is a reasonable smoothness assumption of the curve can also be justified from the property of embedding $H^m(\Omega)$ into $C^j(\Omega)$. Suppose Ω is a subset of \mathbb{R}^n , if $m > j + n/2$, then $H^m(\Omega)$ is embedded in $C^j(\Omega)$. For the surface case [3,4,5,6], the smoothness assumption $u \in H^2$ implies that $u \in C^0$. In the case of curves, $\kappa_t(t), \tau_t(t) \in H^1$ implies that $\kappa_t(t), \tau_t(t) \in C^0$, i.e., the curve has continuous curvature and torsion.

Then the reconstruction of the surface is formulated as the following variational problems. The 2D contour $r_0(t)$ is first backprojected into 3D by minimizing

$$I_1(p(t)) = (p(t), p(t)) \quad (4)$$

If the minimum is reached by $r^*(t) = (x_0^*(t), y_0^*(t), z^*(t))$, then a surface $u(x, y)$ is interpolated by minimizing

$$I_2(u) = \iint_\Omega \left\{ \frac{1}{2} (\Delta u)^2 - (1-\sigma)(u_{xx}u_{yy} - u_{xy}^2) \right\} dx dy \quad (5)$$

with the inhomogeneous Dirichlet boundary condition

$$u|_{\partial\Omega} = g(x, y) = z^* \quad (6)$$

Theorem 1: The energy measure $I_1(p)$ is invariant under linear transformation of the curve, i.e., if $r_2(t) = c r_1(t) + d$, then $I_1(p_1) = I_1(p_2)$.

This theorem is important in the reconstruction because all the similar shapes must have the same energy measure in order for a certain shape to reach the minimum regardless of its size. Consider the energy in an ellipse and a circle and suppose both of them are planar. If the energy measure is defined as

$$E(p(t)) = \frac{k}{2} \int_L \kappa^2(s) ds \quad (7)$$

The energy in a circle is $\frac{k}{2} \frac{2\pi}{R}$; the energy in an ellipse is $\frac{k}{2} f(a, b)$, where a and b are the lengths of the two axes of the ellipse. So an ellipse with larger a and b will have less energy than a circle with a smaller R . This is the reason why by minimizing E , an ellipse cannot be interpreted as a circle [7]. By minimizing $I_1(p)$, an ellipse will be interpreted as a circle [1]. In implementation, $I_1(p)$ is easily discretized as

$$I_1(p(t)) = \frac{k}{2} \sum_{i=1}^N Q_i^2 + \frac{k}{2} \sum_{i=1}^N P_i^2 \quad (8)$$

$$+ \frac{k}{2} \sum_{i=1}^N (Q_{[i+1]} - Q_i)^2 + \frac{k}{2} \sum_{i=1}^N (P_{[i+1]} - P_i)^2$$

where Q_i is the external angle between the two successive sides of the approximating polygon, and P_i is the angle between the normals of the two planes determined by three successive sides of the approximating polygon. Where $[i+1] = (i+1) \bmod(N)$.

Theorem 2: There exists an unique minimum value of $I_1(p)$, for all $p \in B$. Suppose the minimum

is reached by curve $r^*(t)$, then $r^*(t)$ has continuous curvature and torsion.

The problem of minimizing $I_2(u)$ with an inhomogeneous Dirichlet boundary condition can be reformulated as follows. Suppose g is smooth enough, let $g \in H^2$, $v \in H_0^2$, then any $u = v + g \in H^2$ is in the admissible space. The problem becomes finding a $v \in H_0^2$ minimizing

$$I_2(u) = \frac{1}{2} a(u, u) - f(u) \quad (9)$$

$$= \frac{1}{2} a(v, v) - f(v) + a(v, g) + \frac{1}{2} a(g, g) - f(g)$$

with homogeneous boundary condition $v|_{\partial\Omega} = v_n|_{\partial\Omega} = 0$, which is equivalent to $u|_{\partial\Omega} = g$, $u_n|_{\partial\Omega} = g_n|_{\partial\Omega}$. Where $a(u, v)$ is the energy inner product and $a(u, u) = I_2(u)$.

Theorem 3: There exists an unique solution $v \in U$, U is a subspace of H^2 , which minimizes $I_2(u)$ with $u|_{\partial\Omega} = g$.

Next, the general cases are considered. Given boundary and surface contours with piecewise continuous $\kappa_0(t)$. Let the external jump angles of $\kappa(t)$ be α_i , $i=1,2,\dots,n$, the jump angles of $\tau(t)$ at surface discontinuity junctions be β_i , $i=1,2,\dots,m$. Then the reconstruction is formulated as backprojecting the contours into 3D by minimizing

$$J_1(p(t)) = \sum_i I_1(p_i(t)) + \frac{k}{2} \sum_i (\frac{\pi}{2} - \beta_i)^2 + \frac{k}{2} \sum_i \alpha_i^2 \quad (10)$$

where k_β is the orthogonal link factor [1], k_a is the energy factor of curvature jump angles. The term of the jump angles in torsion (torsion jumps across surface discontinuity boundaries) $\sum_i (\frac{\pi}{2} - \beta_i)^2$, is part of the orthogonal links between surfaces based on the principle of maximal orthogonality between surfaces [1, 10]. Suppose $z^*(t)$ is the boundary and $C(x_i, y_i)$ are the points on the surface contours determined by minimizing $J_1(p(t))$. Then the surface is interpolated by minimizing

$$J_2(u) = \iint_{\Omega} \{ \frac{1}{2} (\Delta u)^2 - (1-\sigma)(u_{xx}u_{yy} - u_{xy}^2) \} dx dy \quad (11)$$

$$+ \sum_i \left[\frac{\gamma_i}{2} (u(x_i, y_i) - c(x_i, y_i))^2 \right]$$

$$u|_{\partial\Omega} = g(x, y) = z^*$$

The second term is interpreted as a set of vertical pins scattered inside Ω , the surface is only constrained by attaching ideal springs between those pin tips and the surface (see [3]). Where γ_i is the spring constant.

Again we have the similar results:

Theorem 1': The energy measure $J_1(p)$ is invariant under linear transformation of the curve.

Theorem 2': There exists an unique minimum value of $J_1(p)$, for all $p \in B$. Suppose the minimum is reached by $r^*(t)$, then $r^*(t)$ has piecewise continuous curvature and torsion.

Theorem 3': There exists an unique solution $v \in U$, U is a subspace of H^2 , which minimizes $J_2(u)$ with $u|_{\partial\Omega} = g$.

Now we consider 3D shapes with more than one surface. Barnard [10] recently proposed a maximal orthogonality principle for 3D shape recovery based on psychophysical studies. This principle is modelled by putting ideal springs — orthogonal links — at the corners of the surface discontinuity boundaries. Note that in J_1 the term $\sum_i (\frac{\pi}{2} - \beta_i)^2$ is an

orthogonal link term. So J_1 becomes

$$J_3 = J_1 + \sum_i^k \frac{Y}{2} \left(\frac{\pi}{2} - \gamma_i \right)^2 \quad (12)$$

where k_Y is the orthogonal link factor as in [1].

The finite element method is naturally suited to the problem of surface reconstruction from backprojected 3D contours because of the flexibility in the geometry of the method. Domains of complex shapes, boundary conditions, and nonuniform discretizations of the domain, all of which are features of the backprojected contours, can be easily handled in the finite element method.

Another possibility of dealing with the inhomogeneous Dirichlet boundary condition is by the penalty method. The boundary contours are treated same as surface contours, ideal springs are attached between the contours and the surface at a set of discrete points. Then this becomes a "free boundary" problem, and the solution is only unique upto a linear term, $ax + by + c$. To have a unique solution, there have to exist three noncollinear points on the contours to uniquely determine the linear term [3, 22]. This will always be satisfied in practice. When the spring constraints are strong enough, the solution would be close to the boundary.

For extremal boundaries, the normal to the boundary contours on the x-y plane is the normal to the surface. This can be handled by adding a penalty term to I_2, J_2 .

$$\frac{k_n}{2} \sum_i \left(\frac{\nabla u(x_i, y_i)}{|\nabla u|} \Big|_{\partial\Omega} - n(x_i, y_i) \right)^2 \quad (13)$$

where $n(x_i, y_i)$ are the unit normals to the extremal boundary contour on the x-y plane at points (x_i, y_i) . The constraints are only at discrete points because of the consideration of implementation by the finite element method.

If the boundary consists of partly extremal boundary, partly surface discontinuity boundary, we will have a mixed boundary value problem. It can be treated accordingly.

Using the theory developed, we have proved that an ellipse will be interpreted as a circle, and skewed symmetric figures will be interpreted as real symmetry in 3D. Also several polyhedra and nonplanar polygon shapes have been successfully reconstructed.

Curved 3D shapes have also been successfully reconstructed from 2D contour images. The inputs to the programs are a set of 2D data points obtained by digitizing the contour drawings by a digitizer. $I_1(p)$ or $J_1(p)$ is minimized by the Levenberg-Marquardt algorithm to reconstruct the 3D shapes.

ACKNOWLEDGMENT

This research is supported in part by NIH Grant DE/NS01697-19.

REFERENCES

1. P. Liang, J. S. Todhunter, "Three Dimensional Shape Reconstruction from Minimum Energy Principle", *Proc. of 2nd Intl. Conf. on AI Application*, Miami, Florida, Dec. 1985. (Technical Memo 85-041, Apr. 1985, Pattern Recognition Lab. Univ. of Pittsburgh).
2. P. Liang, J. S. Todhunter, "A Computational Theory of 3D Shape Reconstruction from Image Contours", , Technical Memo 85-091, Sept. 1985, Pattern Recognition Lab., Univ. of Pittsburgh..
3. D. Terzopoulos, "Multilevel Computational Processes for Visual Surfaces Reconstruction", *Compt Vision, Graphics and Image Proc.*, Vol. 24, 1983, pp. 52-96.
4. D. Terzopoulos, "Multilevel Reconstruction of Visual Surfaces: Variational Principles and Finite Element Method Representation", , AI Memo 671, Artificial Intelligence Lab., MIT, 1983.
5. W.E. Grimson, "Surface Consistency Constraints in Vision", *Compt.Vision, Graphics, and Image Processing*, Vol. 24, 1983, pp. 28-51.
6. W.E. Grimson, *From Images to Surfaces: A Computational Study of the Human Early Visual System*, MIT Press, 1981.
7. M. Brady, A. Yuille, "An Extremum Principle for Shape from Contour", *IEEE Trans. on Pattern Analysis & Mach. Intell.*, Vol. PAMI-6, No. 3, May , 1984, pp. 288-301.
8. T. Poggio, "Early Vision: From Computational Structure to Algorithms and Parallel Hardware", *Compt.Vision, Graphics, and Image Processing*, Vol. 31, 1985, pp. 139-155..
9. T. Kanade, "Geometrical Aspects of Interpreting Image as a Three-Dimensional Scene", *Proc.of IEEE*, Vol. 71, No. 7, July , 1983, pp. 789-802.
10. S.T. Barnard, "Choosing a Basis for Perceptual Space", *Compt. Vision, Graphics and Image Processing*, Vol. 29, No. 1, 1985, pp. 87-99.
11. H.G. Barrow, J.M. Tenenbaum, "Interpreting Line Drawings as Three-Dimensional Surfaces", *Artificial Intell.*, Vol. 17, 1981, pp. 75-116.
12. A. P. Witkin, "Recovering Surface Shape and Orientation from Texture", *Artificial Intell.*, Vol. 17, 1981, pp. 17-47.
13. K. Ikeuchi, B. K. P. Horn, "Numerical Shape from Shading and Occluding Boundaries", *Artificial Intell.*, Vol. 17, 1981, pp. 141-184.
14. K. A. Steven, "The Visual Interpretation of Surface Contours", *Artificial Intell.*, Vol. 17, 1981, pp. 47-73.
15. D. Marr, T. Poggio, "A Theory of Human Stereo Vision", *Proc. Roy. Soc. London, B*, Vol. 207, 1980, pp. 187-217.

16. J. E. Mayhew, J. P. Frisby, "Psychological and Computational Studies towards a Theory of Human Stereopsis", *Artificial Intell*, Vol. 17, 1981, pp. 349-385.
17. T. Kanade, "Recovery of Three-Dimensional Shape of an Object from a Single View", *Artificial Intell*, Vol. 17, 1981, pp. 409-460.
18. W. Rudin, *Real and Complex Analysis*, McGraw-Hill, New York, 1974.
19. S. Agmon, *Lectures on Elliptic Boundary Value Problems*, Van Nostrand, Princeton, 1965.
20. P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.
21. R. Courant, D. Hilbert, *Methods of Mathematical Physics*, Interscience, London, Vol. I, 1953.
22. G. Strang, G. J. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, 1973.
23. G. Fairweather, *Finite Element Galerkin Methods for Differential Equations*, Marcel Dekker, New York, 1978.
24. D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Reading, Mass, 1973.
25. L. Collatz, W. Wetterling, *Optimization Problems*, Springer-Verlag, New York, 1975.
26. D. Perkins, "Visual Discrimination between Rectangular and Nonrectangular Parallelopipeds", *Percept. Psychophys*, Vol. 12, No. 5, 1972, pp. 396-400.
27. W.H. Ittelson, *The Ames Demonstration in Perception*, Hafner, New York, 1968.
28. J. Thompson, G. Hunt, *A General Theory of Elastic Stability*, John Wiley & Sons, New York, 1973.
29. R. A. Adams, *Sobolev Spaces*, Academic Press, New York, 1975.
30. S. L. Sobolev, *Applications of Functional Analysis in Mathematical Physics*, Vol. 7, Translation of Mathematical Monographs, American Mathematical Society, 1963.
31. H. Spath, *Spline Algorithms for Curves and Surfaces*, Utilitas Mathematica Publishing, Winnipeg, 1974.

Selection and Use of Image Features for Segmentation of Boundary Images*

Deborah Walters

Department of Computer Science, 226 Bell Hall
University at Buffalo (SUNY), Buffalo, New York, 14260, USA

Abstract

An algorithm is developed to segment arbitrary boundary images into sets of boundaries which represent a single object, and to group together lines which correspond to a single object or object part. The algorithm is based on features which were found to be used by humans in the early stages of visual processing, and which have a high correlation with perceptually significant aspects of images. In addition, the data structure used is based on the image representation used in the primate visual cortex.

By using perceptually valid features, the algorithm is able to enhance the perceptually significant edges in an image using simple, local, parallel computations. It demonstrates that selective processing can occur in the parallel stages of early visual processing, without domain specific knowledge, iterative processing, or top-down control of some mechanism to shift attention.

KEYWORDS: image segmentation, boundary images, visual psychophysics

*This project was supported by grant IST 8409827 from NSF.

1. Introduction

Images contain too much information for humans or machines to process all of it in detail. The human visual system solves this problem by performing a initial, cursory analysis of the entire image which allows it to pick out automatically what is important in the image (Treisman, 1986), and then to selectively process that information in preference to the rest of the information. That is, a rapid, parallel analysis of the entire image indicates which regions are likely to contain the most useful information. Then the following stages of analysis, which require more focused, serial processing can concentrate primarily on the preselected regions. This can also be a useful approach for a computer vision system, and in fact the parallel computation of intrinsic images can be viewed as an example of the first stage (Barrow and Tenenbaum, 1981). This paper describes another type of processing which enables the early visual processing to indicate which regions of an image are likely to contain the most useful information, and to selectively process such regions in parallel. This is accomplished by selectively processing image features which have a high correlation with perceptually significant aspects of an image. This is not a new approach. For example, edge detection techniques pick out object boundaries and other edges which are more perceptually significant than more uniform image regions (Marr, 1982). However, the research in this paper presents a new set of features which can enable strong inferences about which regions of an image contain the most perceptually relevant information.

Determining which aspects or features of an image contain the most useful information, and should therefore be preferentially processed is difficult because there are nearly an infinite number of potential features. The dimensions of physics cannot

necessarily be used to determine which features are relevant, as perceptual features may lie along some other dimensions. For example, the perceived color of a region depends not only on the wavelength and intensity of the light reflected from it, but also on the relative contrast between it and neighboring regions. So how can perceptually relevant features be found?

The question is further complicated as one set of features may be ideal for one task, but useless for another. One basic machine vision task is to segment an image into different regions which correspond to different objects, or object parts. This may be possible based on the color, shading, texture and shape information. But, are the color and shading information necessary for image segmentation? Not always, as humans can readily segment simple line drawings or boundary images which lack that information. So one way to study image segmentation is to study line drawing perception, and as line drawings are much simpler than natural images, this should make the selection of features easier. Once features are found from line drawings, then it is possible to test them in the analysis of natural images.

But even for simple line drawings, it is not obvious which features should be used. As the goal is to find perceptually significant aspects of an image, and then to determine which features correlate with those aspects, it is desirable to determine what aspects of an image have perceptual significance for humans. It is not possible to just introspect about possible features, as the relevant preattentive stages of human visual processing are not available for conscious introspection (Julesz & Schumer, 1981). The approach taken in this paper is to use psychophysical experiments to explore preattentive vision and to discover image features used by humans. Once potential features are found, their usefulness is tested by developing a computational algorithm based on them, and then testing the algorithm. The algorithm developed here can segment arbitrary boundary images containing both straight lines and curves. It is a simple, data-driven, bottom-up approach, which requires no domain specific knowledge, and demonstrates the importance of using perceptually valid features.

2. Psychophysical Experiments

The psychophysical experiments are based on the perceived contrast of lines phenomenon (Walters and Weistein, 1982a). The patterns in Fig. 1 can be used to illustrate this phenomenon. When viewed at low contrast the lines in the cube (Fig. 1a) appear to have higher contrast than the lines in Fig. 1b. If these differences in perceived contrast can be correlated with the presence of particular image features, it would suggest that stimuli with those features are processed differently from stimuli lacking the features. In particular, stimuli having features associated with high perceived contrast may be preferentially processed. The aim of the psychophysical experiments was to isolate such features. The experiments have been reported elsewhere (Walters and Weistein, 1982b; Walters, 1984, 1985), so only a brief description is included here.

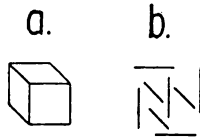


Figure 1

By looking at lots of pairs of patterns designed to differ in terms of various global and local properties, it was found that the difference in perceived contrast did not correlate with any of the global features. For example, closure, global connectivity, perceived 3-dimensionality and objectness did not correlate with perceived contrast. Local features were also explored, and the presence of angles, and the number of free line ends were ruled out. The only two features that did correlate with perceived contrast were line length, and the local connections between the ends of the line segments. For lines which subtended less than one degree of visual arc, perceived contrast was a positive function of line length. For longer lines, there was no correlation between perceived contrast and length. The other local property is the way in which line ends are connected, and experiments show that there is actually a hierarchy of end connections. Figure 2 shows the results of one such experiment. The brightness of various patterns formed of 30 minute line segments was measured relative to a line which subtended 60 minute of visual arc. Some of these patterns could be referred to as the "L", "Fork", and "T" junctions from the Huffman-Clowes tradition (Huffman, 1971; Clowes, 1971; Waltz, 1975). But it turns out that that is not the most useful classification. As section 3.2 explains, it is better to classify these patterns in terms of the spatial relations between the ends of the lines.

From the results in Fig. 2 we can see that line segments joined at their ends have higher contrast than segments where one end abuts the middle of the other segment. And these abutting lines have higher contrast than lines that intersect, while intersecting lines have higher contrast than unconnected lines.

Further experiments found one additional pattern in the hierarchy, as shown in Fig. 3. Two lines connected end-to-end (pattern A) have higher contrast than three lines connected end-to-end (pattern B), which have higher contrast than lines which connect end-to-middle (pattern C), which have higher contrast than the lines which intersect, which in turn have higher contrast than parallel lines.

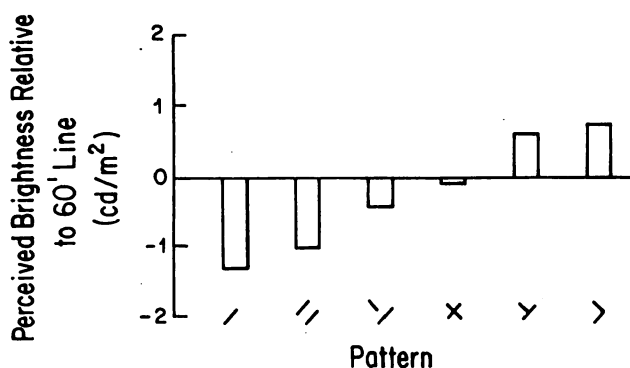


Figure 2

3. Computer Model of Contrast Enhancement

The psychophysical experiments provide evidence that the length of lines, and the connections between the ends of lines, are basic features for human vision. This hypothesis was further tested by implementing it as a computer model. The model receives a boundary image as its input, and outputs the "perceived" contrast of the pattern.

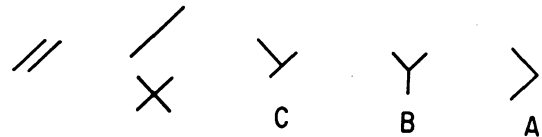


Figure 3

3.1. Enhancement Rules

The model uses the presence of the different types of end connections to implement the following enhancement rules. 1) Each section of line with length L is enhanced by amount "1". 2) Lines terminating at Type A connections are enhanced by amount "a". 3) Lines terminating at Type B connections are enhanced by amount "b". 4) Lines terminating at Type C connections are enhanced by amount "c". 5) Amount "a" > amount "b" > amount "c".

3.2. Detection of Features

The presence of the features can be detected in an image by defining the different end connections in terms of a discrete geometry (Rosenfeld, 1979). The first step is to determine whether each point in the image is part of a line. Thus points can be labeled as either non-line or line points. The line points can then be further broken down into end points and non-end (middle) points. But consider the intersection of two lines. The point that lies on the intersection can be considered to be a middle point of one or the other line, or can even be considered an end point of each of four shorter line segments. Thus some way of defining the point is needed which avoids these ambiguities. An edge detection technique could be used to label each point in the image with the orientation and amplitude of the best line or edge centered on that point. But at the intersection point, it is not so clear what the best line would be. Some edge detection techniques give the orientation of either one or the other line, while others give an average of the two orientations. So, just at a point that provides lots of information about the scene, the edge detection methods don't give sensible answers.

Another problem in edge detection arises because many of the popular approaches to edge detection in computer vision are based on the use of the mathematics of continuous functions. This creates problems in detecting the Type A connection, which is defined in terms of a tangent discontinuity, as in the mathematics of continuous functions, discontinuities are problematical. Poggio et al. (1985) have suggested that the solution to this problem is to regularize the computation. For example, get rid of the discontinuities by convolving the image with a gaussian, and then look for edges in the blurred image. The advantage of this technique is that patterns can then be represented as smooth continuous functions, but it is rather unfortunate from the contrast enhancement point of view, as it gets rid of the tangent discontinuities, which appear to be such important features for early vision. So edge representation methods based on continuous functions are not very useful for this model.

The solution to these edge representation problems can be found by looking at how edges are represented in the primate visual cortex. If an amplitude/orientation scheme were used, there would need to be two "edge" neurones in the primary visual cortex for each retinal ganglion cell: one to signal the amplitude of the edge at that point, and another to signal the orientation of the edge. But the cortex does not have that organization, instead there is a whole column of edge neurones for each spatial location, and each neurone is sensitive to edges with a narrow range of orientations (Hubel and Weisel, 1968). So, instead of just signaling the "best" amplitude and orientation, the orientation column signals the amplitude at many orientations. This representation has several advantages over the amplitude/orientation scheme. For example, at the intersection of two lines, both orientations can be represented, which would disambiguate the pattern.

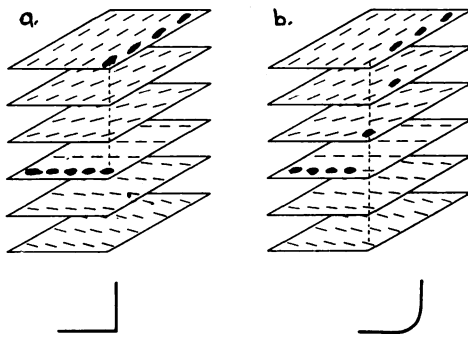


Figure 4

The data structure used in the general enhancement algorithm, is based on the orientation representation of the mammalian visual system. Figure 4(a) shows the basic form of the data structure. It is an orientation plane representation. It is a 3-d space where each point represents a short piece of line having a specific orientation and located at a specific x-y location. An image is transformed into this representation by convolving the image with a separate oriented edge kernel for each orientation plane. It is possible to construct any number of separate orientation planes for this representation. In the current implementation 8 or 16 orientation planes are used. In addition, if boundaries are present at different scales in the image, then a separate orientation plane structure is needed for each scale. This would be required for grey-level images, though not for line drawings containing a single width of line.

One advantage of the orientation plane representation is that it makes it easy to define lines and find tangent discontinuities. A line is defined to be a set of dark pixels such that each pixel is connected to neighboring pixels of similar orientation. The specific definition of 'connected' is orientation dependent. Line pixels can only be connected to other line pixels which lie within a certain x,y distance, and a certain orientation distance, and the greater the x,y distance, the greater the possible orientation distance which can yield a connection. These definitions can be used to label all the pixels in an image as either nonlinear, end 'e' or middle 'm' points.

Connections can be defined in terms of these 'e' and 'm' labels. For example, for a Type A connection located at point (x1,y1), examining the x1,y1 position in each plane would yield exactly two 'e' labels, and no others. (Actually, the examination may involve a small neighborhood around the (x1,y1) point.) This suggests how to define the different connections in terms of the 'e' and 'm' labels.

3.3. Completeness of the Feature Set

Another question that needs answering is, are these features geometrically complete? Does it cover the space of all possible connections? This question can be answered in terms of all the possible combinations of 'e' and 'm' labels. Figure 5 shows all the possible connections for straight lines of just three possible orientations, in terms of the number of 'e' and 'm' labels at the center of the connections. The upper left three are not connections. The upper right two are both intersections, which receive no contrast enhancement. Three of the connections were used in the psychophysical experiments, and there are two additional connections needed to cover the space. The new connections are hypothesized to belong to the classes as labeled.

There are only a limited number of orientations in Fig. 5. For more orientations, the set would be extended, and the labels can also be extended. Everything out to infinity in the top row is a Type D connection. Everything out to infinity in the second row is a Type C connection. And everything out to infinity in the other rows is a Type B connection. It is important to be able

	Number of Colinear Arms			
	0	1	2	3
Number of Noncolinear Arms	0	/	X D	X D
1	-	/	X C	X C
2	└	X B		
3	Y B			

Figure 5

to label all possible types of end connections, but at the same time, the probability of any of the higher order types occurring in a natural scene are very small, thus they are not as important as the few seen in Fig. 5. It can now be seen that the set of connections is complete, and that there is a means of detecting the presence of the features as all connections can be classified into the four perceptually valid classes using these rules. 1) All connections with exactly two 'e' labels are of type A. 2) All connections with two 'e' labels, and at least one additional 'e' or 'm' label are of type B. 3) All connections with exactly one 'e' label and one or more 'm' labels are of type C. 4) All connections with no 'e' labels and two or more 'm' labels are of type D.

3.4. Dealing with Curved Boundaries

The examples thus far have dealt only with line drawings containing straight lines. To be useful the algorithm should be able to deal with curves as well. Does the perceived contrast of curves also vary with the type of end connections present? Further psychophysical experiments confirmed this hypothesis. Figure 6 shows the hierarchy of end connections for curved lines, with the lines on the right having the highest perceived contrast, and the lines on the left having the lowest. The results for the curved lines are identical to the results for the straight line segments, although curved lines have a possibility of one further type of end connection, as shown in the A' pattern. With curved segments, two segments can merge or join into one, without a discontinuity of the tangent. So, one additional rule must added to the algorithm to deal with such connections.

It is easy to see the relation between these curved connections and the straight line connections, but what is the relation between a Type A connection, and the same pattern with the discontinuity slightly smoothed, as in Fig. 4? The principle of stability argues for continuity of interpretation when small perturbations are made in an image. Thus when the right angle of is perturbed to form the smoothed angle, the interpretation should remain similar. In order to have the same orientation plane interpretation a means of defining the end connections of curved lines is needed. The solution is that end connections occur when a line passes through J orientation planes within K pixels; J and K are variables which determine the sensitivity to curves.

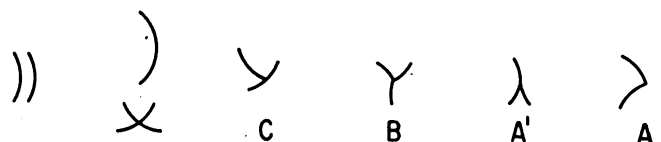


Figure 6

3.5. Model Implementation

The model can be implemented in parallel. Imagine a simple processor at each pixel in an image, such that each processor receives input only from a small neighborhood of pixels. Each processor can compute whether a particular spatial relation between the ends of lines is centered in its neighborhood, and if so, can send the appropriate enhancement out along the appropriate pixels. Note that it's not the mechanism of contrast enhancement that is being modeled - it's the overall computation that is of concern.

4. Model Results

Figure 7a shows the output of the computer model for four of the psychophysical patterns. The results are displayed in terms of a threshold. The highest threshold - that is the highest contrast lines are at the top. The threshold becomes lower in each subsequent line. At the bottom is the lowest threshold where all of the lines which were present in the patterns appear.

The model results agree with the experimental results for all of the patterns used in the psychophysical experiments. Thus the hypothesis that perceived contrast is a function of line length and the type of connections between the ends of lines, is further supported.

4.1. Uses of Features Suggested by the Model

A further use of the computer model is to go beyond the psychophysical results. One limitation with the human experiments is that subjects are only able to make global judgements of

contrast - they can say pattern A was overall brighter than pattern B, but they cannot say whether a particular line in pattern A was brighter than the others. But the computer model gives such results, which can help in determining why such processing might be useful.

Actually, some of the possible uses can be seen with these simple stimuli of Fig 7a. Lines which are part of object contours are enhanced relative to lines which form texture, or are perceived as noise. And, the outer contours of objects are enhanced relative to the inner contours.

Looking at another example (Fig. 7b) shows how the model goes one step further. Figure 7b contains two distinct objects, one partially occluding the other. At the highest threshold the lines composing the two objects are not spatially continuous. The model selectively enhances objects in the foreground, and helps to group lines into two sets which correspond to the two objects.

Figure 7c shows the model results for an impossible object. For the possible object, the model results at an early level (level 2) show the main properties of the object - a blob with a hole in it. That is, the model is giving the topological structure of the object at a very early level. But with the impossible object, at the first level it is represented as a single object, then as one object with two sub-objects, or object components. The model again agrees with our perception of one object if we look at one corner of the drawing, and another if we look at the diagonally opposite corner, and neither we nor the model can get the perceptions to merge.

5. Uses of Image Features

The results of the computer model give more support to the hypothesis that the length of lines, and the spatial relations between the ends of lines, are perceptually valid features. But how should these features be USED?

5.1. Current Approaches

Various theories concerning the use of features have been proposed in computer vision. One conceptually simple use of features is to represent objects in terms of a list of features (Feldman, 1985). A model of an object can be expressed in terms of features and the relations between them, and then portions of an image can be compared to the model to see if the object is present. This is similar to the way line drawing junctions were first used by Roberts (1965). But this use involves domain specific knowledge, which is a major drawback as it is thus not easily extendible to deal with arbitrary images.

Guzman(1979), Kanade (1981), Draper (1981), and Lee et al.(1985) have used very similar line drawing features in their boundary image interpretation algorithms. The features are used in various constraint satisfaction systems. This paper presents another, related use of end connection features, which is not limited to trihedral vertices, and accomplishes a somewhat different task.

5.2. Selective Enhancement

A different use of features is suggested by the psychophysical experiments and computer model. Lines appear to be selectively enhanced based on the presense of a few basic features. (The potential usefulness of this enhancement is described in the next section.) It appears that selective enhancement is possible, even in the automatic parallel stages of processing. This requires no top-down processing, no domain-specific knowledge, and no iterative processing.

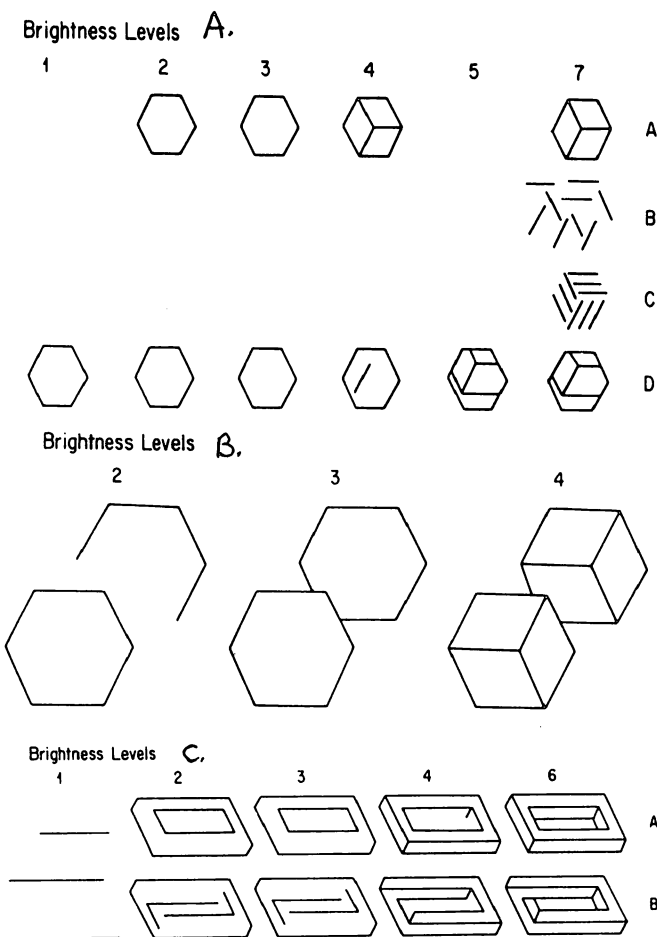


Figure 7

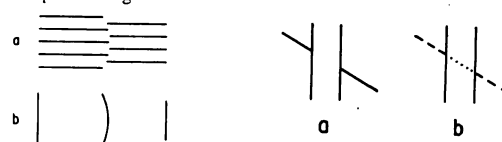


Figure 8

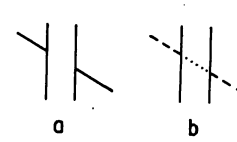


Figure 9

6. Perceptual Significance of Selective Enhancement

Why should such parallel selective enhancement be useful? The computer model provided some hints. The outer contours of objects were enhanced more than the inner contours, and object contours were enhanced more than lines interpreted as texture or noise, and the highest contrast lines were correctly segmented. But why would these results be helpful?

The selective enhancement of outer contours is important for object recognition. An object can usually be recognized just from its silhouette, which is simply its outer boundary - outer contours have a special perceptual significance. And the edges of a silhouette can only contain type A connections. This may be the reason that end-to-end connections appear to receive the most contrast enhancement. And this supposed correlation between type A connections and the outer contours of objects makes it possible to infer that the most enhanced lines in an image have a high probability of having arisen from the occluding contours of objects.

The type B connections can arise from either an inner or outer contour of an object, and thus do not have as strong a correlation with outer object boundaries. Even when we divide the type B connections into forks and arrows as Chakraverty and others do (Chakraverty, 1979; Lee et al., 1985), they can still both arise from both types of object contours.

Yet, if two simple assumptions are made, both type A and type B junctions have a high correlation with object contours in general.

The first assumption is:

Assumption 1: Viewing position is representative.

(This means we assume we are looking at an object along a viewing direction which is not one of the few viewing directions which results in the accidental alignment of object boundaries or wires in a scene. (Binford, 1981; Cowie, 1982))

Result 1: Two or more lines meeting at a junction should be interpreted as two or more wires or object boundaries that meet.

Assumption 2: Object position is representative.

(This means we assume objects or wires in a scene are not accidentally aligned. The first assumption concerns looking at objects in such a way as to make them appear to be accidentally aligned. The second assumption concerns cases where the objects are in some form of accidental alignment with each other, independent of the viewing position.)

Result 2a: Line ends meeting at a point should all be interpreted as having arisen from the same object.

Result 2b: Two or more line middles falling on a point should be interpreted as wires or texture boundaries.

Result 2c: Connections containing both ends and middles are most generally interpreted as object boundaries that either occlude or meet other object boundaries.

Result 2d: The end line in a connection should be interpreted as arising from a different object from the middle lines.

The assumptions about nonaccidental alignment do not mean that images with accidental alignment cannot be enhanced or segmented using this algorithm. It just means that the most general interpretation for a basic feature will be utilized. Thus in the majority of cases, the correct interpretation will arise, while a few cases may exist where the algorithm gives an incorrect interpretation.

From these assumptions we see that in Type A and B connections the lines have a high probability of having arisen from the same object. This result makes these connections the most useful for grouping together lines which correspond to a single object or object part. This result is also useful for segmenting the image into sets of lines which represent a single object. Note that this type of segmentation is different from Richards' segmentation of curves (Richards & Hoffman, 1983). His work shows how to segment the curves of an object into sets which correspond to various object parts, but not how to segment an entire image into different objects.

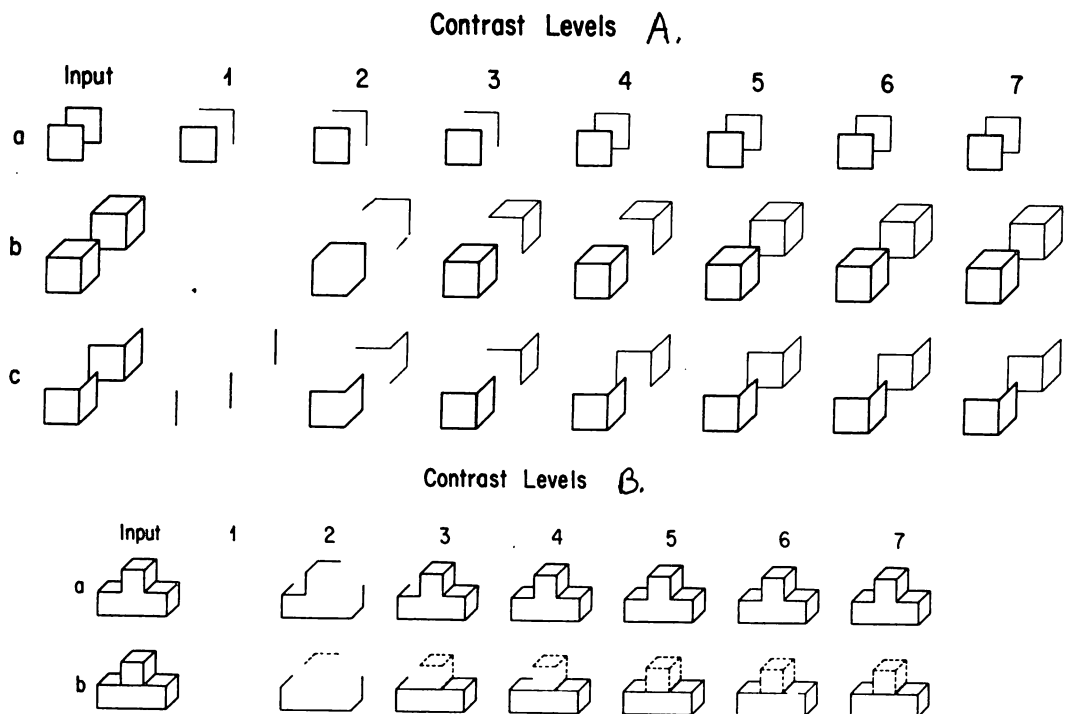


Figure 10

So, from the contrast enhanced lines, certain inferences about the line drawing can be made. Why would these inferences be useful for a visual system? Well, as previously

mentioned, a major problem for a visual system is that there is too much information in a visual image to process all of it in detail. One solution is to have some automatic preprocessing system which determines which lines or areas contain the most important information, and then to concentrate the serial processing on those areas, while ignoring other potentially less fruitful areas. This model automatically enhances those lines which have a high probability of being part of object contours, rather than just part of texture or noise. If the next stage of processing has to be selective, it can "attend" only to the enhanced lines and thus not waste resources processing spurious edges. But note that some stages of the selective processing can be done in parallel, and they do not require the top-down control of some mechanism to shift attention (Ullman, 1986).

7. Additional Perceptual Effects

Before describing the results of implementing the enhancement algorithm for curves and straight lines, a couple of other perceptual effects incorporated into the algorithm need to be described. The connections between the ends of straight and curved lines are basic features for the human visual system. The human visual system is also sensitive to virtual edges such as the ones seen in Fig. 8a. Thus the enhancement algorithm may be improved by including the ability to deal with virtual edges. Again the orientation plane representation makes it easy to represent virtual edges, by allowing edges to grow perpendicularly from the ends of lines. This is similar to the boundary contour completion of (Grossberg and Mingolla, 1986).

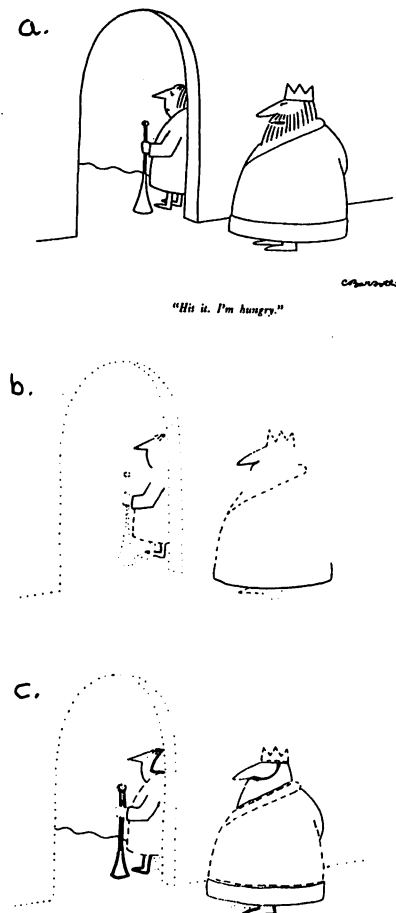


Figure 11

Figure 9 demonstrates one further trick of the human visual system, which is incorporated into the algorithm. The figure is perceived as representing a single diagonal line, which appears to pass behind the surface represented by the vertical lines. Thus pairs of Type C connections, when aligned and having a particular symmetry can be interpreted as occlusion of a single line. A related approach has been used by (Lee et al., 1985) to find hidden vertices in line drawings.

8. Segmentation Examples

The contrast enhancement algorithm can be implemented using the orientation plane representation. Figure 10a shows some examples of applying the segmentation algorithm to 2-D, origami, and 3-D objects. The drawings are correctly segmented in all three cases, as indicated by the different line styles for the different objects. Again the outer contours of the objects are enhanced more than the inner contours, and objects in the foreground are enhanced relative to occluded objects. Note that the one set of rules can deal with the three separate domains.

Another example is seen in Fig. 10b, where the first and second images differ only by a single line segment, yet the second alone is represented as two separate objects. Note that this segmentation is indicated early in the process - ie. at Level 2. Thus even at this early stage the segmentation is correct. Note that the segmentation is performed without using either implicit or explicit models of objects, and the top-down processing that model matching requires. This is different from most current algorithms, and enables any boundary image to be processed.

9. Grouping Performance on an Arbitrary Line Drawing

To demonstrate the ability of the enhancement and grouping algorithm, to deal with an arbitrary line drawing, a cartoon from the New Yorker was processed in accord with the algorithm. Figure 11a shows the original cartoon. In Fig. 11b, only the most enhanced groups of lines are displayed - those involved in Type A and A' connections. Only 61 of the total of 86 lines are present, yet object recognition is possible. If just the remaining 25 lines are displayed, object perception is not possible - which is weak evidence that the algorithm picks out the most perceptually salient lines.

The grouping at this stage is depicted by the different line styles. Sixteen of the twenty-three separate objects or object parts are represented at this stage. (Due to reproduction limitations, only four line styles are used in the figure, however each instance of line style indicates a separate set of lines.) Again the algorithm is effective in reducing the complexity of the drawing in terms of the number of lines, without diminishing the grouping capabilities.

Figure 11c shows the final grouping of the cartoon. The sets all correspond to object or object parts that are readily named by humans: ie 'crown', 'robe', 'cuff', 'sleeve', 'foot', etc. There are no groups which would have to be described as "the upper right hand portion of object x", which again suggests that the grouping has perceptual significance. The algorithm could be used as a powerful preprocessor for a scene analysis system, as it accomplishes a lot, given just a handful of simple rules. Later stages of analysis could use the enhanced sets of lines as input to an object recognition algorithm (Pentland, 1985; Biederman, 1985).

10. Texture Boundaries

Up to this point the spatial relations between the ends of curves and lines have been discussed, which are one type of boundary found in images. But the algorithm could equally well be applied to other boundaries - for example, texture boundaries. Instead of using lines or edges as the input to the algorithm, the boundaries defined by differences in texture could be contrast enhanced in accordance with the spatial relations between their ends. This is an interesting example, because it may be that the end-connections are used twice in this type of analysis. Julesz has

many impressive experiments designed to uncover the features used by humans in texture segregation, which he calls textons (Julesz & Bergen, 1983). At last count, the texture features include color, elongated blobs (which includes lines), their free terminators, and crossings. Julesz' rejection of global properties such as closure, and global connectivity for preattentive texture discrimination agrees with my findings that these properties are not relevant to changes in perceived contrast. And as there is a strong correlation between the number of free terminators, and the type of end-connections, it may be that the hierarchy of end connections found to alter perceived contrast, can equally well explain texture segregation of patterns composed of lines and curves.

11. Natural Images

A technique for finding texture boundaries in natural images is necessary before the selective enhancement algorithm can deal with natural images which contain regions defined by texture edges rather than intensity edges. But can the selective enhancement algorithm work for natural images which do not contain texture boundaries? We are currently addressing this question by implementing the algorithm for natural images.

There are two basic problems when applying line drawing algorithms to natural images. First is the problem of extracting the edges from the image, leaving the connection information intact. We are developing edge detection techniques, similar to those of Canny (1984), to alleviate this problem. A related problem is that many spurious edges are found with many edge detection techniques. The lack of contrast enhancement for short edges, and the selective enhancement of end connected edges both reduce such problems.

The other problem is that extracted edges are often noisy and contain large gaps. But a similar problem is found in many cartoons - end connections or lines are implied but not explicitly present. The selection enhancement algorithm uses the creation of virtual edges to solve this type of problem, and it may be possible to extend this solution to natural images.

12. Conclusion

The end connections of lines and curves appear to be basic features which allow bottom-up processing of boundary images using a single set of simple rules. The contrast enhancement algorithm suggests that certain selective processing can be performed in the parallel stages of preattentive processing. It is a data-driven approach which accomplishes tasks previously thought to require domain specific knowledge. Object models are obviously necessary for some stages of object recognition, but the contrast enhancement algorithm demonstrates that some steps which were previously thought to require model matching, do not. This shows how productive bottom-up processing can be when psychophysically valid features are used in perceptually valid ways.

13. References

- Barrow, H.G. & Tenenbaum, J.M., Interpreting line drawings as three-dimensional surfaces. *Artificial Intelligence*, 1981, 17, 75-116.
- Biederman, Irving, Human image understanding: Recent research and a theory, *Computer Vision, Graphics and Image Processing*, (in press).
- Binford, T.O., Inferring surfaces from images. *Artificial Intelligence*, 1981, 17, 205-244.
- Canny, John F., Finding Edges and Lines in Images. Master's thesis, MIT, June, 1983.
- Chakravarty, I. A generalized line and junction labeling scheme with applications to scene analysis, *IEEE Trans. Pattern Anal. Machine Intell* 1, 1979, pp202-205.
- Clowes, M.B., On seeing things. *Artificial Intelligence*, 1971, 2, 79-116.
- Cowie, R., Modelling people's interpretation of line drawings. D. Phil. Thesis, University of Sussex, 1982.
- Draper, S.W., The use of gradient and dual space in line-drawing interpretation. *Artificial Intelligence*, 1981, 17, 461-508.
- Feldman, J.A., Four frames suffice: A provisional model of vision and space. *Behav. and Brain Sci.* 8, 2, 1985.
- Grossberg, S. and Mingolla, E., Neural dynamics of form perception: Boundary completion, illusory figures, and neon color spreading. *Psych. Rev.* (in press).
- Guzman, A., Decomposition of a visual scene into three-dimensional bodies. In A. Grasseli (Ed.), *Automatic Interpretation and Classification of Images*. New York: Academic Press, 1969.
- Hubel, D.H. & Wiesel, T.N., Receptive fields and functional architecture of monkey striate cortex. *J. of Physiology*, 1968.
- Huffman, D.A., Impossible objects as nonsense sentences. In B. Meltzer & D. Michie (Eds.) *Machine Intelligence*, 6 pp 295-323, Edinburgh: Edinburgh University Press, 1971.
- Julesz, B. & Schumier, R.A., Early visual perception. *Annual Review of Psychology*, 1981, 32, 575-627.
- Julesz, B. & Bergen, J.R., Textons, the fundamental elements in preattentive vision and perception of textures. *Bell System Technical Journal*, 62, 1619-1646, 1983.
- Kanade, T., Recovery of the three-dimensional shape of an object from a single view. *Artificial Intelligence*, 1981, 17, 409-46.
- Lee, S.H., Haralick, R.M. & Zhang, M.C., Understanding objects with curved surfaces from a single perspective view of boundaries. *AI*, 26, 1985, pp 145-169.
- Marr, David., *Vision*. San Francisco: W.H. Freeman, 1982.
- Pentland, Alex. P., Perceptual organization and the representation of natural form, *SRI Technical Note 357*, 1985.
- Poggio, Tomaso, Torre, Vincent & Koch, Christof, Computational vision and regularization theory, *Nature*, 317, p 314 - 319, 1985.
- Richards, W. & Hoffman, D.D., Codon constraints on closed 2d shapes. A.I. Memo No. 769, MIT, 1984.
- Roberts, L.G., Machine perception of three-dimensional objects. In J.P. Tippet et al. (Eds.) *Optical and Electro-optical Informa*
- Rosenfeld, A., *Picture Languages*, (New York: Academic Press), 1979.
- Treisman, A.M., Preattentive processing in vision. *Comp. Vis. Graph. & Imag. Proc.* (in Press).
- Ullman, S., Visual routines: Where bottom-up and top-down processing meet. in Schwab & Nusbaum, eds, *Pattern Recognition: Visual Perception*, Academic Press: New York, (in press).
- Walters, D.K.W. & Weisstein, N., Perceived brightness is influenced by structure of line drawings? *Invest. Ophthalm. & Vis. Sci.*, 1982, 22, 124.
- Walters, D.K.W. & Weisstein, N., Perceived brightness is a function of line length and perceived connectivity. *Bulletin of the Psychonomic Society*, Sept. 1982, 130.
- Walters, D.K.W., Local connections in line drawing perception: A computational model. *Investigative Ophthalmology and Visual Science*, 1984, 25, 200.
- Walters, D.K.W., The use of natural constraints in image segmentation. *Proc. of Int. Soc. for Optical Engin.*, vol 548, 1985, pp 27-34.
- Waltz, D.L., Understanding line drawings of scenes with shadows. In P.H. Winston (Ed.), *The Psychology of Computer Vision*. New York: McGraw Hill, 1975, pp 19-22.

Cortical Representation of Texture Primitives

A.E.J. Walford and M.E. Jernigan
Department of Systems Design Engineering
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

Abstract

The apparent spatial frequency and orientation organization of the primary visual cortex is used as a basis for texture description. A Frequency, Orientation, neural firing Rate, and spatial Phase (FORP) representation is proposed for the analysis of natural textures and the synthesis of test textures. Higher order texture analysis such as discrimination and segmentation in this FORP space is discussed.

Keywords: texture analysis, vision models, primary visual cortex models

Introduction

Many researchers have tried to duplicate the ability of the human visual system to segment natural scenes based on the different texturing of surfaces. We choose to define texture as that property of surfaces that can be described by the local pattern of spatial variation of intensity. We also take as different textures those that can be differentiated by human perception. In this light, we base our analysis of texture on a model of the early human visual system. In particular, we choose a model of the primary visual cortex since both orientation and spatial frequency information exist here.¹

We use this information in an orthogonal feature extraction space to generate simple test textures and analyse natural textures.

Texture and the Visual Cortex

Given that the primary visual cortex is well suited to texture computations, is there any evidence that it actually performs them? The following evidence indicates that this could indeed be true.

First of all, Julesz² mentions the high speed discrimination ability of human subjects. This implies that texture discrimination is a low level visual function and must be early in the visual chain. Kimchi and

Palmer,³ through the use of perceptual experiments found that texture is processed separately in the visual system from shape and structure. Lastly, Berlucchi and Sprague⁴ use lesion studies to deduce that shape and structure encoding does not exist in the primary visual cortex. They suggest the primary visual cortex could be used for texture analysis.

The primary visual cortex appears well suited for texture analysis, it appears to actually perform texture computations, and it can use this information to improve image segmentation.

A Visual Cortex Model

Pollen and Ronner¹ describe a model of the primary visual cortex which outlines various functions of cortical neurons. These functions include the retinotopic spatial map, ocular dominance, orientation, spatial frequency and spatial phase. Hubel and Wiesel⁵ first described a small separate processing region in the primary visual cortex as the *hypercolumn*. The *hypercolumn* was responsible for analysing a small area of the visual field for orientation information. The hypercolumn has become the name for the 0.5 mm wide cortical area that contains orientation and frequency selectivity neurons for a small visual region. There exist hypercolumn regions to cover all of the visual field.

Our model of the primary visual cortex is based on this hypercolumn structure. One hypercolumn is modeled as a three dimensional space. The space consists of a spatial frequency axis, an orientation axis, and a spatial phase axis. Each point in this space corresponds to a neuron that is selective to a particular set of frequency, orientation, and phase. A magnitude component is also included in this space to account for the strength of response to this particular set. The magnitude is coded by the neuron as a neural firing rate. This model is a modified two dimensional Fourier space with all symmetric regions removed. The model has been named FORP (spatial Frequency, Orientation, neural firing Rate, and spatial Phase).

The FORP based tools that were developed allowed us to both synthesize and analyse textures. One of the tools allowed a window of a texture image to be chosen and be represented in FORP space. The other allowed the experimenter to place individual points or group of points in the space and then have the texture generated that corresponded to this pattern of firing neurons. Figures 1 thru 6 are outputs of these programs.

Figure 1 shows a representation of FORP space and the corresponding synthesized texture. It shows the three axes of frequency (range of 0 to $F_S/2$), orientation (range of 0° to 180°), and neural firing rate (normalized to a range of 0 to 1). There is a grid on the frequency/orientation plane, and the intersection of two grid lines corresponds to a neuron or a set of neurons. The intensity of the bar at a frequency/orientation point represents the phase. The lowest intensity corresponds to a phase of 0 and the highest to a phase of 2π . If there were a set of phase sensitive neurons then the bar's intensity would represent which phase neuron in this set was responding maximally. The image beside the graph contains a homogeneous texture generated from the FORP space data. The little square in the bottom left corner of the image delineates the local region that is represented by the FORP space. The remainder of the texture image is a mosaic-like repetition of this small subimage.

To demonstrate the nature of FORP space we have performed a number of simple texture syntheses. In Figure 1, a single point is placed at a frequency of $F_S/16$ (that is 1/16th of the spatial sampling frequency), and an orientation of 110° (note that 0° is vertical). The resulting synthesized texture is a spatial sinusoid at 110° and has 2 periods in its 32 by 32 sub-window. This point is moved along the frequency and orientation axes (Figure 2). Note that the frequency change modifies the sinusoidal spacing and the orientation change modifies the sinusoid's angle.

To answer the question, "What will natural textures look like in FORP space?", we performed some texture analyses shown in Figures 3 thru 6. Images of the natural textures oriental rattan,⁶ diatom,⁷ and J-Cloth¹⁰ were each sampled with windows at different positions. The window size was 64 by 64 taken out of a 256 by 256 image, except the diatom samples which were done with a 32 by 32 window. The most important aspect of these FORP representations is their similarity in shape for the same texture, and their dissimilarity for different textures. In Figures 3 and 4, both FORP space representations of oriental rattan look quite similar. The majority of neural response is along the 0° , 90° and 180° lines. This is understandable when one realizes that oriental rattan is composed mostly of horizontal and vertical edges. In Figure 5 the FORP space of diatom

looks quite different from that of rattan. The neural responses are spread quite evenly in orientation. This is to be expected of a texture that is composed of circles, since a circle has edges of all orientations. Again, in Figure 6 the FORP space of J-Cloth¹⁰ is quite different from previous ones.

D'Astous⁸ comments on this similarity of frequency domain representations and its importance: "... the power spectrum is fairly invariant to minor changes in structure caused by either low magnitude additive noise, or by small deviations in the periodicity of the texture. This is of particular relevance to the problem of discriminating natural textures which tend to be noisy and, though many textures exhibit regularity to a certain extent, are not strictly periodic."[†]

Required Accuracy

The accuracy of the FORP parameters was studied to see how it affected the representation of textures. A window of a texture was converted to FORP space at various frequency and orientation accuracies. This representation was then used to generate a texture which was compared to the original.

The amount of orientation information required depended on the type of texture. A synthetic texture with only horizontal and vertical edges required only 2 levels of orientation whereas the diatom texture required at the least 18 levels (each 10° wide). It was felt that 20 or more orientation levels would be sufficient for most textures.

The number of spatial frequencies was changed by modifying the size of the discrete frequency transform. In all cases the accuracy of the result was not affected but different amounts of the textures were captured. For a few spatial frequencies only a small piece of the texture would appear in the mosaic, and for many frequencies a large area of the original would appear.

Does the above relate to the accuracy of the primary visual cortex? Hubel and Wiesel⁹ found the accuracy of the orientation neurons to be approximately 10° . But there is a major difference between the FORP model orientation sensitivities and those of the primary visual cortex. The orientation sensitivities of neurons overlap much like the spatial frequency sensitivities. It can be shown that this may improve the actual orientation accuracy in the hypercolumn by quite a bit. In terms of frequency, nothing is lost as long as the local analysis region size changes with frequency accuracy. It has been shown that the receptive field sizes in the primary visual cortex are indeed proportional to their frequency bandwidth.¹

[†] D'Astous 1983, p. 55

A model of the hypercolumn in the primary visual cortex that has many of hypercolumn traits has been presented. The model seems to be useful for characterizing texture but more texture samples need to be analysed.

Application of the Visual Cortex Model

Our visual system, if given the information contained in FORP space, could extract further information to characterize different textures. Neurons could be connected to these frequency and orientation selective neurons in such a way as to extract texture based features.

Michael¹⁰ has described the method of inter-neuron information transfer. The output of a nerve cell is transmitted along its axon and is then connected to the input of another nerve cell via a chemical junction called a synapse. The synapses can either inhibit the nerve cell or excite it. The receiving neuron performs a type of summation or integration of all its synaptic input which results in an overall excitation level. If this excitation is above a threshold then the neuron will fire and transmit its excitation to other connected neurons. If the total inhibitory input is larger than the total excitatory input then the neuron will be inhibited from firing. Each synapse can also have an associated weight or a multiplication factor that emphasizes or de-emphasizes certain neural inputs.

This structure of inter-neuron connection is an effective means of constructing pattern recognizers. When a certain pattern of excitations is present on the axons of the input neurons, the processing neuron can recognize it and fire proportional to the strength of that pattern. In the context of the FORP cortical model, higher level neurons will be able to recognize certain patterns of firings of the frequency and orientation neurons, and as much as these patterns correspond to texture, these neurons will recognize textures.

Figure 7 depicts a simple realization of such a pattern recognizer. The hypercolumn is represented by a grid of small boxes. Each box corresponds to a neuron that is sensitive to a particular spatial frequency and orientation in one local area of the visual field. These neurons output to feature detector neurons through inhibitor and excitor synapses. The feature neurons shown can recognize very simple patterns in the FORP space, and may also receive either inhibitory or excitatory input from neighbouring hypercolumns. In Figure 7 the *Feature 1* neuron will detect a pattern of all one orientation with a strong low frequency component, a weak second frequency component, and a strong third frequency component. The *Feature 2* neuron will detect a pattern of one strong high frequency component and weak components on adjacent frequency and orientation neurons. These particular patterns might correspond to a particular texture or to a particular characteristic of

textures.

Looking at Figures 3, 5, and 6 which are the FORP space representations of oriental rattan, diatom and J-ClothTM respectively, it is easy to see how the neural mechanism described above could pick out patterns characteristic of each of these textures. Many more neural connections and some careful choice of inhibitory and excitatory synapse weights would be needed. A feature detector neuron could be designed to detect each of these textures since they are so different in FORP space. The implication is not that the primary visual cortex actually has a neuron for every type of texture, instead the suggestion is that a computer or electrical realization of such a mechanism could have separate texture recognizers if that was the end goal. In the visual cortex, the first level of feature extraction neurons would deal with texture attributes instead of specific textures. Second level neurons could combine these texture attributes to further specify texture. This hierarchical organization would be more general and more flexible than having textures recognized at the first levels.

One potential problem with this feature extraction mechanism is its sensitivity to orientation. A feature neuron that fires for a texture at one orientation may not fire when presented the same texture at a different orientation. People have little difficulty identifying the oriental rattan texture regardless of its rotation. Perhaps our features should be orientation invariant.

Jolicoeur¹¹ demonstrates through perceptual experiment that rotational invariance need not exist in low level visual processing. He states, "... this experiment reveals a clear-cut effect of orientation on identification time. ... These results allow us to argue against a general model of pattern recognition based solely on the extraction of 'orientation-invariant features'."†

If texture is to be used in segmentation then orientation differences might be useful. Consider a cube with identically textured surfaces. When viewed, the main difference between the texture of the cube faces will be orientation, and hence orientation differences will play a major role in segmentation of the cube faces. Given these two insights, it would appear that these low level texture features need not be orientation invariant.

Segmentation

This feature extraction model can be extended to perform rudimentary texture based segmentation. Imagine the output of these feature cells being coded into a grey level depending on which cells were maximally excited. Then place these grey levels in a retinotopic map and generate a form of two dimensional image. Segmenting this image using grey level techniques such as thresholds,

† Jolicoeur 1985, p. 293

region growing, or edge detection is equivalent to segmenting the original visual scene by texture. The most probable candidate in human vision is segmentation via edge detection and shape recognition. Edge detection mechanisms that exist at this cortical level could be very similar to those found in the retina. An edge at this level will correspond to an edge between differently texture regions.

A combination of intensity edge maps, texture edge maps, colour differences, motion detection, and binocular depth cues can all be used in separating objects and performing image segmentation. Marr¹² discusses the use of multiple visual processes in providing accurate and stable decisions about surfaces. Each process involved in segmentation would provide its best information on surface boundaries, but only when all evidence is studied and judged for strength and weakness will the segmentation process decide on the final separation of constituent objects.

Summary

The image segmentation computer of the future will need to use various processes to mimic human success. It will use intensity and texture edges, colour differences as well as motion and depth cues. We have proposed a model of the primary visual cortex in which texture discrimination and segmentation can be performed. Demonstrations of the model indicate that similar textures are represented similarly and dissimilar textures dissimilarly. With extensions of further neural processing levels it may be possible to construct an efficient and effective texture segmentation processor.

References

1. Pollen, D.A. and Ronner, S.F., Visual Cortical Neurons as Localized Spatial Frequency Filters, *IEEE Trans. on Systems Man and Cybernetics*, pp. 907-916 (September/October 1983).
2. Julesz, B., Textons, the elements of texture perception, and their interactions, *Nature* **290** p. 91 (March 1981).
3. Kimchi, R. and Palmer, S., Separability and Integrality of Global and Local Levels of Hierarchical Patterns, *Human Perception and Performance* **11**(6) pp. 673-688 (December 1985).
4. Berlucchi, G. and Sprague, J.M., The Cerebral Cortex in Visual Learning and Memory, and in Interhemispheric Transfer in the Cat, pp. 415-440 in *The Organization of the Cerebral Cortex*, ed. Schmitt, Worden, Adelman and Dennis, MIT Press (1981).
5. Hubel, D.H. and Wiesel, T.N., Uniformity of Monkey Striate Cortex: A Parallel Relationship between Field Size, Scatter and Magnification Factor, *Journal of Comparative Neurology* **158** pp. 295-306 (November/December 1974).
6. Brodatz, P., *Textures: A Photographic Album for Artists and Designers*, Reinhold, New York (1968).
7. Wolberg, L.R., *Micro-Art*, Harry N. Abrams Inc., New York (1970).
8. D'Astous, F.T., Textural Feature Extraction in the Spatial Frequency Domain, Ph.D. Dissertation, University of Waterloo (1983).
9. Hubel, D.H. and Wiesel, T.N., Sequence Regularity and Geometry of Orientation Columns in the Monkey Striate Cortex, *Journal of Comparative Neurology* **158** pp. 267-293 (November/December 1974).
10. Michael, C.R., Retinal Processing of Visual Images, *Scientific American*, (May 1969).
11. Jolicoeur, P., The time to name disoriented natural objects, *Memory and Cognition* **13**(4) pp. 289-303 (July 1985).
12. Marr, D., *Vision*, W.H. Freeman and Company, San Francisco (1982).

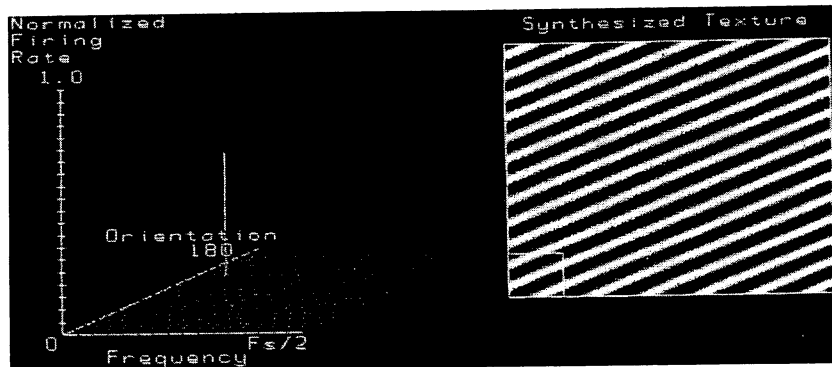


Figure 1 : Single Point in FORP Space



Figure 2 : Point Moved in Orientation and Frequency

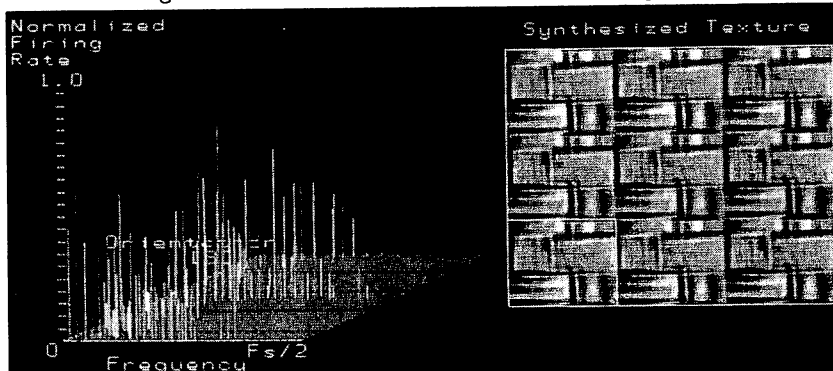


Figure 3 : Oriental Rattan - FORP Representation

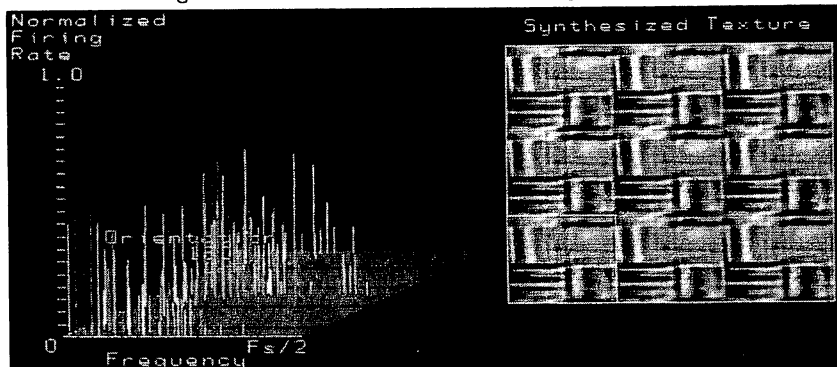


Figure 4 : Shifted Oriental Rattan - FORP Representation

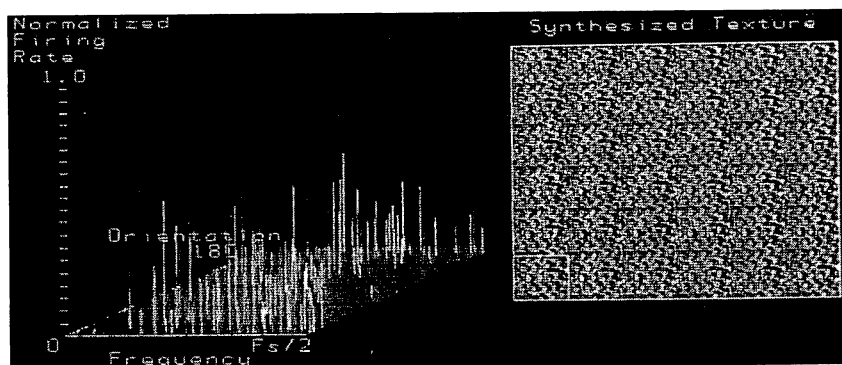


Figure 5 : Diatom Texture - FORP Representation

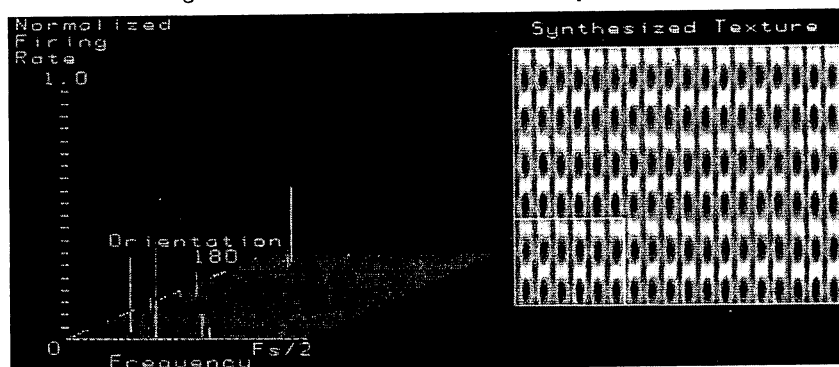


Figure 6 : J-Cloth® Texture - FORP Representation

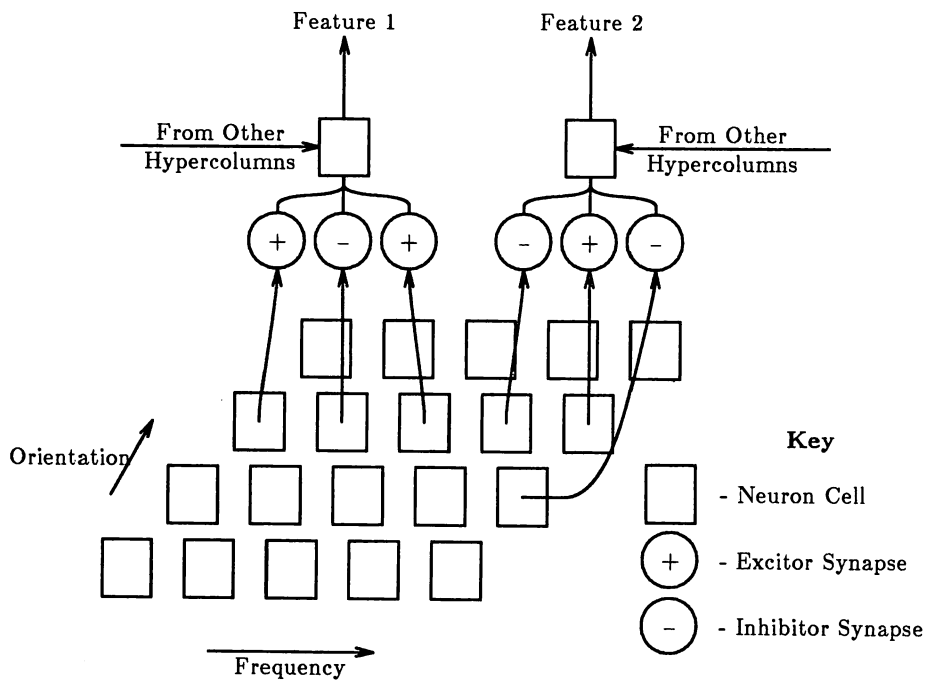


Figure 7 : Simplified Model of Hypercolumn Feature Extraction

SPEEDED PHASE DISCRIMINATION :
EVIDENCE FOR GLOBAL TO LOCAL PROCESSING

J.M. Barr
Experimental Psychology Department
University of Oxford. U.K.

Abstract

In recent years much attention has been given to the advantages of multiple resolution pre-processing methods in computer vision. There is strong evidence that the parallel extraction of luminance changes over different spatial scales also occurs in human visual perception. The first experiment confirms the strong evidence that responses to low resolution signals can be elicited as much as 100 msec faster than to high spatial frequency stimuli of the same contrast. A further experiment measured the reaction time to discriminate the relative phase of the higher frequency component of a luminance grating comprising a fundamental and its second harmonic. It was found that the decision can be made more rapidly when the fundamental is low than high frequency. On the assumption that the gross structure of spatial forms is conveyed by low spatial frequencies, this supports the idea that the substrate for the subjective impression that rough descriptions of visual forms precede detailed perception. is the progressive increase in response time with frequency, of the visual mechanisms implementing spatial filtering.

Current Address :
THORN EMI Central Research Laboratories,
Hayes, Middlesex, U.K.

Key Words :

Visual Perception, Phase Discrimination,
Multi-resolution Image Processing

Introduction

Within most natural scenes, intensity changes occur over a range of spatial scales, so it has been suggested that a general purpose vision system may require some form of early representation that captures and makes this explicit. Over the past decade a number of multi-resolution schemes have been described in the image processing literature. [1,2,3,4,5,6].

Over the same period and independently, considerable empirical evidence has been amassed from both psychological and neurophysiological research, supporting the idea that a form of multiple resolution representation is employed in the early processing stages of biological visual systems. It is asserted that at each point in the retina there exist several contrast sensitive mechanisms each detecting luminance changes over a different spatial scale. As spatial scale is equivalent to spatial frequency (for signals containing a single frequency component) the neural mechanisms can be described as filters in the spatial frequency domain. The simplest model of the filter impulse response is the difference of two circularly symmetric Gaussian functions. The output of a set of filters tuned to the same frequency, covering the entire retina is termed a channel, and is equivalent to the parallel convolution of the image with a single operator. Each channel is assumed to act independently of any other, hence this processing scheme is known as the multiple independent channels model [7].

The independent channels model has been remarkably successful in predicting contrast thresholds in a variety of laboratory experiments, and in generating a great deal of further research attempting to specify the spatial filter characteristics. However little attention has been given to their organization, function or utility. A description of the properties of

individual mechanisms is insufficient to explain the fundamental problems of shape description and object recognition.

Early papers proposed that the hypothesised neural filters implement spectral analysis. If phase information is discarded it would then be possible to perform pattern recognition with translational invariance, and if reference templates are stored and matched in terms of the ratio of frequencies, with size invariance [8,9]. However there are several objections to this notion. Spatial phase is of cardinal importance to the visual system. It is still possible to recognise objects if amplitude information is discarded by equalizing all components, provided phase information is preserved [10]. The one octave bandwidth of the filters is too great to permit high resolution spectral analysis, and furthermore, while Fourier Transform based methods may permit the recognition of simple shapes presented in isolation, it would be much more difficult to achieve more complex tasks such as scene segmentation in the frequency domain.

Marr and Hildreth [11] propose a space domain function for spatial frequency filters, in which at each image location, mechanisms tuned to different frequencies provide independent evidence of luminance changes over different scales. This information is then integrated in local oriented edge detector units. Critical problems with this scheme are the alignment of the output from different operators, its performance on blurred edges and its performance in the presence of noise.

In this paper the bank of spatial filters will be considered as a multi-resolution pre-processing front end, which provides a rich description of luminance changes over a range of spatial scales, upon which higher level interpretive processes may operate.

An advantage of pyramidal processing schemes in computer vision is the possibility of information flow in several directions within the data structure [12]. Projection operations permit information acquired at low resolution to guide processing at higher levels. For example in matching applications it seems an efficient strategy to rapidly discard the maximum number of incorrect alternatives by first matching at a coarse level of resolution, saving the more computationally intensive high resolution process for a reduced set of alternatives. Similarly in shape analysis it is more efficient to first

locate the approximate boundaries of the form with a coarse analysis before focussing local operations on optimal regions. Data flow in the opposite direction involves the integration of high resolution information and its reduction to lower levels, eg. block quantization. Lateral processing is restricted to a single level. It is not known whether there are such interactions between the levels of the multiple resolution structure of the human visual system but it is well established that the temporal properties of low spatial frequencies channels differ from those tuned to higher spatial frequencies in that the former mechanisms exhibit a greater sensitivity to temporal transients [14]. Low spatial frequency mechanisms behave as derivative operators to temporal changes in contrast while high spatial frequency mechanisms operate as temporal integrators. Hence the possibility is raised that the output of each spatial frequency channel is produced asynchronously.

Intuitively it appears that on first glance of a scene, we obtain an immediate rough impression of the approximate forms, locations and extents of the principle objects. Perception rapidly becomes more detailed over time [14]. This form of global precedence can be interpreted in terms of the independent channels model as the progressive acquisition of representations of increasing spatial frequency. Immediately after stimulus presentation only the blurred output of low spatial frequency filters is available, and over a fraction of a second the image representation is sharpened by the addition of higher frequency components. It is not known whether this phenomenon has any utility in terms of neural implementations of projective multi-resolution algorithms, or whether it is merely a processing bottleneck, a dysfunctional epiphenomenon.

The idea of asynchronous parallel channels has several interesting consequences, two of which will be addressed empirically in this paper. The first is that the visual detection of stimuli containing lower spatial frequencies should be faster than of stimuli containing only higher frequencies. Secondly, visual discrimination should be faster if the stimuli differ in their low spatial frequency content than if they only differ in their high frequency content.

Experiment 1

The first experiment measures simple reaction time (sRT) to the presentation of sine wave gratings, replicating a

result obtained by Breitmeyer [15]. The stimulus is the abrupt appearance of a luminance grating displayed on a CRT screen. The observer's task is to indicate his detection of the change from a blank to a luminance modulated screen by pressing a microswitch as quickly as possible.

A very simple model of the subject's performance in the reaction time task comprises two components, a sensory, and a motor stage. On the abrupt presentation of a visual stimulus it is assumed that only those spatial filters tuned to the stimulus produce a sensory response, and that their response magnitude follows some growth function, increasing over a short duration following stimulus onset. The decision to press the microswitch is taken when the perturbation of the sensory output function exceeds a criterion value. The duration of motor response is assumed to be constant (approx. 100 msec), independent of sensory factors including the spatial frequency of the stimulus.

The total sRT is the sum of the durations of these two stages. Any variation in the empirically obtained latencies with stimulus spatial frequency can therefore be entirely attributed to differences in the time taken for the outputs of the spatial filters to exceed some criterion level. The stimulus is a one dimensional luminance function sinusoidally modulated about a mean level. This is spread vertically on the screen by a high frequency oscillator to give the appearance of a vertical grating.

Results for one subject at two contrast levels are shown in Figure 1. It should be possible to obtain estimates of sensory latency from the sRT values by subtracting from them the constant motor time. It can be seen that there is indeed an increase in the time course of response with increasing stimulus frequency. A response to the detection of a low contrast 9 cpd grating is not made until up to 100 msec after that to a 0.5 cpd stimulus of the same contrast. It can also be seen that reaction time decreases with increasing stimulus contrast and the extent of the spatial frequency based time difference diminishes. On the basis of the assumptions made above, this can be taken to imply that spatial filters tuned to higher frequencies have longer latencies than those tuned to lower frequencies.

Experiment 2

Having confirmed the evidence for temporal asynchronies between spatial frequency channels in a detection task, it was decided to investigate whether

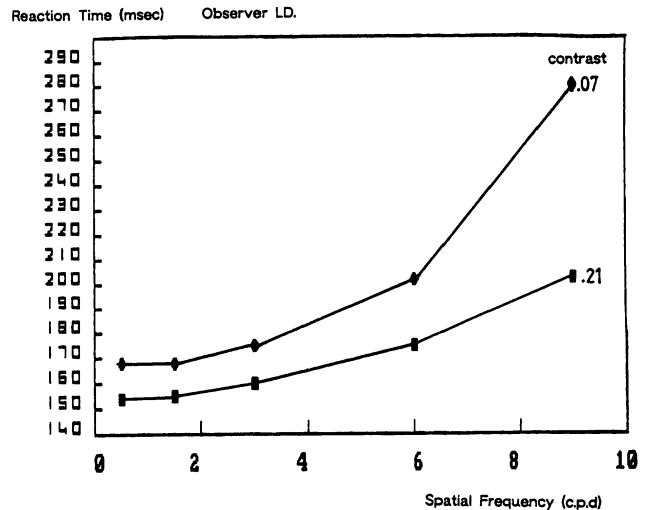


Figure 1. Reaction Time to sinusoidal gratings for one subject at 2 contrast levels

the effect transfers to discrimination. Most natural visual stimuli do not comprise a single frequency component but complex spectra. Furthermore it is the phase rather than the amplitude spectrum that determines shape recognition [10]. Therefore a choice Reaction Time (cRT) task was chosen requiring the discrimination of the relative phase of a two component compound grating. If channel asynchrony imposes deterministic delays on the transmission of information then decisions based on higher frequency information should be delayed relative to those based on low spatial frequency information.

The stimuli used in the experiment were the first two sinusoidal components of a square wave, added in either square wave or triangle wave phase. The luminance at each point, $L(x)$ is given by Equation 2:

$$L(x) = L_m + c \sin(2\pi f x + \Phi_1) + c/3 \sin(2\pi 3f x + \Phi_2) \quad (\text{Equation 2})$$

where L_m is mean luminance, and c, f and Φ are contrast, frequency and phase respectively. Their luminance profiles are shown in Figure 2. Observers had to identify each stimulus as rapidly as possible. The model for the performance of the cRT task is similar to that for sRT except that a discrimination process must intervene between sensory detection and the initiation of the motor response. A motor response can only be initiated after the detection of the third harmonic and the discrimination of its phase relative to the fundamental. Assuming the speed of phase discrimination is constant with spatial frequency, it should not be possible to make the discrimination until the higher frequency information has reached the

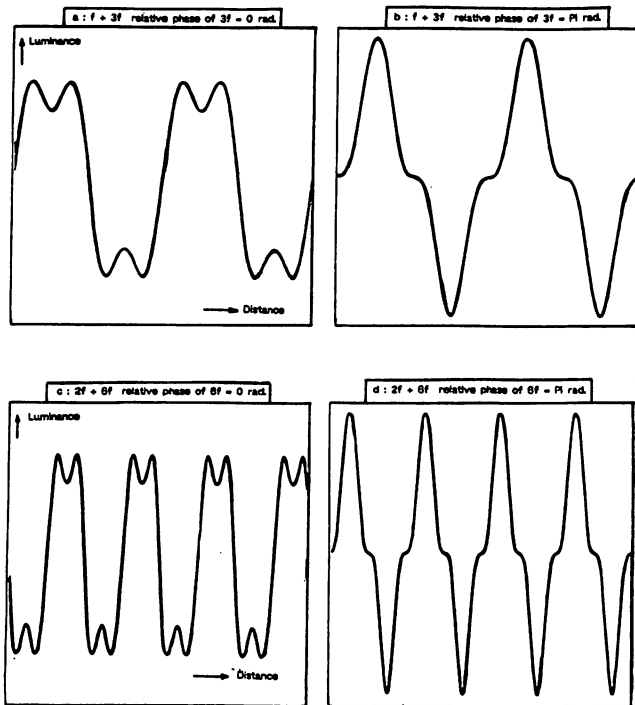


Figure 2 Examples of the luminance profiles of the stimuli used in Experiment 2.

decision mechanism. Therefore there should be an increase in choice reaction time with frequency that should depend solely on sensory delay and should increase at the same rate as reaction time to detect the high frequency component presented alone.

Each stimulus was the sum of two sinusoids separated in frequency by a factor of three. In each block of trials the same fundamental frequency was always presented, but the relative phase of the harmonic randomly varied between either 0 or π on each trial with equal probability. The subject was given two response keys, one assigned to each alternative, and instructed to indicate the perceived phase relationship as rapidly as possible by pressing the appropriate switch. The subjects were also presented with the third harmonic stimuli in isolation in a SRT task.

Figure 3 shows the results for 4 subjects. The increase in SRT with the frequency of the third harmonic stimuli follows the pattern of Experiment 1. The results for the CRT task are plotted in terms of the frequency of the third harmonic. Phase discrimination for the low frequency compound stimuli appears to take approximately 80 msec longer than detection of its slower component. If this reflected the additional complexity of the phase discrimination decision then reaction time to the high

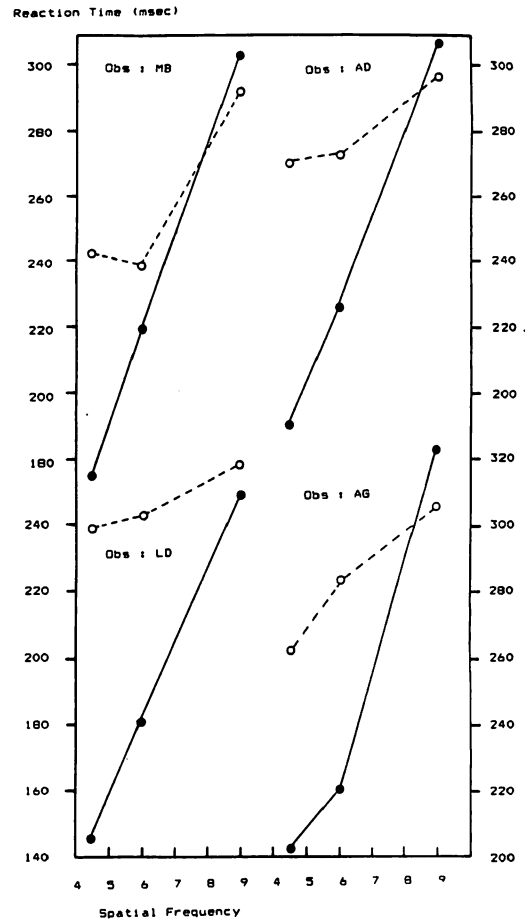


Figure 3. Results of Experiment 2 illustrating Reaction Time as a function of Spatial Frequency for 4 subjects.

Closed circles represent mean simple Reaction Times to single component gratings.

Open circles represent mean choice Reaction Times for correct responses in the speeded phase discrimination task. The symbols are plotted against the frequency of the third harmonic component of the compound stimulus.

frequency compound would be expected to show a similar additional delay. However the surprising result is that the the relative phase of the third harmonic can be discriminated as rapidly as the detection of the same stimulus presented in isolation.

It appears that the slope of the CRT function is more nearly parallel to that of the SRT function if plotted in terms of the frequency of the fundamental, rather than the third harmonic. The shallower slope follows the shallower increase in SRT with frequency at lower spatial frequencies and higher contrasts. One possible explanation for the anomalous result of Experiment 2, therefore is that the performance of the task is based not upon discrimination of relative phase but upon discrimination of the peak to peak amplitudes of the

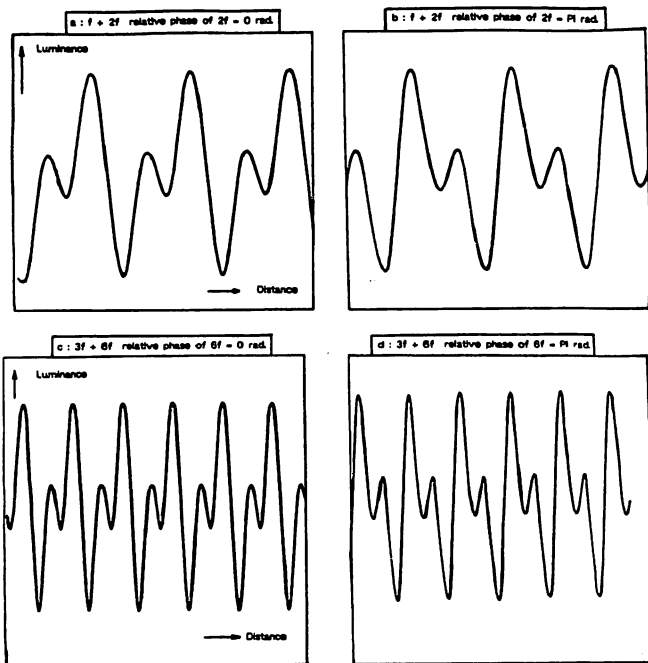


Figure 4 Examples of the luminance profiles of the stimuli used in Experiment 3.

waveforms. Although the contrasts of the two frequency components are the same in both phase relationships, the ratio of the difference of image maxima and minima between the square wave and triangle wave is 1.41. So subjects could have been responding to the stimulus on the basis of global contrast rather than relative phase. Therefore it was decided to repeat the experiment with a condition in which peak to peak contrast is controlled.

Experiment 3

In this experiment the stimulus is a compound of a fundamental and a second harmonic of equal contrast. The luminance at each point is given by Equation 3 :

$$L(x) = L_m + c \sin(2\pi f x + \phi_1) + c \sin(2\pi 2f x + \phi_1) \quad (\text{Equation 3})$$

The luminance profile of stimuli in the two phase relationships and two fundamental frequencies are shown in Figure 4. It can be seen that global contrast cannot be used as a cue as the stimuli are related by a reflection. SRT to the second harmonic stimuli presented in isolation was also measured.

Reaction Time (msec)

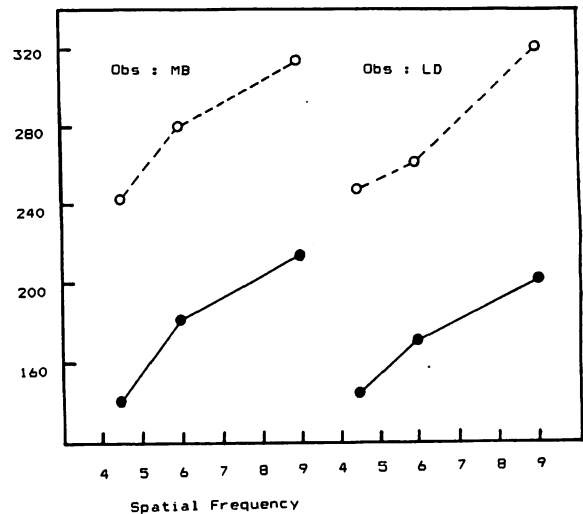


Figure 5. Results of Experiment 3 illustrating Reaction Time as a function of Spatial Frequency for 2 subjects.

Closed circles represent mean simple Reaction Times to single component gratings.

Open circles represent mean choice Reaction Times for correct responses in the speeded phase discrimination task. The symbols are plotted against the frequency of the second harmonic component of the compound stimulus.

The results for two subjects are shown in Fig 5. The increase in cRT is parallel to the increase in sRT with spatial frequency. Thus when global contrast is removed as a cue there does indeed appear to be an increase in the time taken to perform a discrimination task when the decision has to be based upon the high rather than low spatial frequency content of the stimulus.

Discussion

Experiment 1 provided strong evidence for the idea of asynchronous detection of spatial frequency components when presented in isolation. Low frequency stimuli can be detected up to 100 msec faster than high frequency stimuli of the same contrast. That contrast also increases detection time further supports the idea of global precedence, as the amplitude spectra of most natural stimuli are low pass. Thus the idea of asynchronous channel operation can be viewed as providing a computationally explicit explanation of the intuitive observation of global precedence.

Assuming it is possible to predict the detectability of any stimulus at a constant mean luminance given a knowledge of the frequency spectrum of the stimulus and the observer's modulation transfer function, it should

similarly be possible to predict visual conspicuity and the growth of detail perception over the first few moments of inspection of any static visual stimulus, from a knowledge of its Fourier spectrum and the observer's reaction time performance. The results of Experiment 3 are also encouraging, illustrating that frequency related detection latency differences transfer to a discrimination task.

However the analogy between computer vision and human multi-resolution systems is far from perfect. One major problem is raised by the results of Experiment 2. While there is good evidence for individual spatial frequency components being detected independently at threshold contrast, the evidence is not compelling for suprathreshold stimuli. Assuming that the channels have a bandwidth of approximately 1 octave it should not be possible to integrate energy from two frequency components separated by a factor of 3. That this appears to have occurred in Experiment 2, enabling global contrast to be used as a cue in the discrimination task, implies that channel bandwidth in some way increases with stimulus contrast. If this is so it seriously weakens the idea of independent channels.

References

- [1] Rosenfeld, A. (1980) Multi-resolution Image Processing and Analysis. Springer-Verlag, Berlin, FRG.
- [2] Hall, E.L., Wong, R.Y., and Rouge, J. (1977) Sequential Scene Matching with Hierarchical Search. Proc. IEEE Southeast Conference, Williamsburg, Va., U.S.A. pp.402-405.
- [3] Witkin, A.P. (1984) Scale Space Filtering : A New Approach to Multi-scale Description. ICASSP 39A.1.1-4.
- [4] Baker, K.D. & Sullivan, G.D. (1980) Multiple Bandpass Filters in Image Image Processing. IEE Proc. Vol 127 Part E(5) pp.173-184.
- [5] Burt, P.J. & Adelson, E.H. (1983) The Laplacian Pyramid as a Compact Image Code. IEEE Trans. Communications Vol COM-31(4) pp.532-540.

[6] Crowley, J.L. & Parker, A.C. A Representation for Shape Based on Peaks and Troughs in the Difference of Low-pass Transform. IEEE Trans Pattern Analysis and Machine Intelligence Vol PAMI-6(2) pp.156-170.

[7] Campbell, F.W. & Robson, J.G. (1968) Application of Fourier Analysis to the Visibility of Gratings. J. Physiology (London) Vol 197, pp.551-566.

[8] Blakemore, C. & Campbell, F.W. (1969) On the Existence of Neurones Selectively Sensitive to the Orientation and Size of Retinal Images. J. Physiology (London) Vol 203 pp.237-260.

[9] Pollen, D.A., Lee, J.R. & Taylor, J.H. (1971) How Does the Striate Cortex Begin the Reconstruction of the Visual World ? Science Vol 173, pp.74-77.

[10] Piotrowski, L.N. & Campbell, F.W. (1982) A Demonstration of the Visual Importance and Flexibility of Spatial Frequency Amplitude and Phase. Perception Vol. 11 pp.337-346.

[11] Marr, D. & Hildreth, E. (1980) Theory of Edge Detection. Proc. Roy. Soc. London B, Vol. 207 pp.187-217.

[12] Hanson, A.R. & Riseman, E.M. (1980) Processing Cones : A Computational Structure for Image Analysis. In Structured Computer Vision by Tanimoto, S. & Klinger, A. Academic Press, New York.

[13] Robson, J.G. (1966) Spatial and Temporal contrast sensitivity functions of the Visual System. J. Opt. Soc. Am. Vol. 56, pp.1141-1142.

[14] Navon, D. (1977) Forest before Trees : The Precedence of Global Features in Visual Perception. Cognitive Psychology, Vol 9, pp.353-383.

[15] Breitmeyer, B.G. (1975) Simple Reaction Time as a measure of the Temporal Response Properties of Transient and Sustained Channels. Vision Research Vol 15, pp.1411-1412.

The work reported here was supported by an S.E.R.C. research studentship.

CORRESPONDENCE IN APPARENT MOTION: DEFINING THE HEURISTICS

MARC GREEN

PSYCHOLOGY DEPARTMENT
YORK UNIVERSITY
NORTH YORK, ONTARIO M3J 1P3

ABSTRACT

Correspondence matching in apparent motion is based on two heuristics: match images if they 1) have a similar form and 2) are in close proximity. Psychophysical experiments are used to define these heuristics. Observers judged motion path between images in a competition paradigm. Results showed that the tokens used in form matching are spatial frequency and orientation. Further, proximity is defined in a 3-D spatial reconstruction rather than 2-D retinal coordinates. A possible representation for the computation of correspondence is a multidimensional detector space, with dimensions including spatial frequency, orientation, X, Y and Z (or disparity) coordinates.

INTRODUCTION

A remarkable property of biological vision systems is the ability to deduce that two images, seen at different places and/or times, represent the same physical object. The advantages of this property are nicely exemplified in the phenomenon of apparent motion: when viewing a series of static pictures, or "frames", each object in one frame moves to the location of the corresponding object in the subsequent frame. Coherent motion is perceived only if the visual system, after considering successive frames, can properly match images corresponding to the same object. This "correspondence problem" is presumably solved by application of heuristics to provide a "preference metric" (13) which evaluates the affinity between potential matches. Preference metrics can be derived by two general classes of heuristic: 1) match images of similar form and 2) match images with the greatest spatial proximity. At first glance, this recipe for matching seems simple enough, but real difficulties arise when implementation is attempted. These heuristics need to be more precisely defined by answering the following questions. First, what form primitives are used as tokens in correspondence matching? Second, is proximity defined in two-dimensional retinal coordinates or in an internal, 3-D reconstruction of space. This question has important implications since use of a 3-D metric requires that a depth must be assigned each form token before matching can proceed. The studies

described below are addressed to answer each of these questions.

THE FORM HEURISTIC

The notion that correspondence matching is based partly on form similarity has been around for a long time. However, it has proved surprisingly difficult to identify correspondence tokens since apparent motion tends to be independent of form similarity. Early studies (8, 14) found that when there was only one image in each frame, the apparent motion seen with two identical images was readily perceived with two different images. The first would deform gradually into the second with no loss of motion continuity. More recently experimenters (3,11) have used competition methods and have likewise concluded that form similarity plays no role in token matching.

Why has it proved so difficult to identify correspondence tokens? Two possible explanations come to mind. First, stimuli were either geometric forms, circles, squares, letters, etc. or alphabetic characters, which differ in high spatial frequency content, but are similar in low spatial frequencies. Both human psychophysical (1) and computer (9) experiments have resulted in the view that one representational stage in early visual processing is the activity in arrays of detectors which are sensitive to edges at different resolution. At each resolution level, the detectors are activated only by a narrow band of spatial frequencies. Geometric shapes and alphabetic characters would all stimulate similar populations of coarse, low resolutions detectors. If activity in detectors at different resolution were tokens, then there would be strong affinity between all such images. Second, previous investigators have used a flash technique which produced a luminance transient (i.e., a D. C. offset) accompanying the presentation of form. The luminance flux per se might be used as a token for matching. This seems plausible because it has been suggested (4) that the visual system contains two parallel sets of detectors for analyzing spatio-temporal luminance change. The detectors are modeled as difference of Gaussians (DOG's) with different temporal properties. If the inhibitory Gaussian is developed simultaneously with the excitatory, then the detector is "sustained" and is highly tuned to aspects of form such as spatial

frequency and orientation. If inhibition is delayed, the detector is "transient", responds to D. C. flux and shows little selectivity to form. Correspondence might be determined by matching patterns of activity in these transient detectors.

I tested these possibilities in a set of psychophysical experiments (5). The initial assumption was that correspondence matching is mediated by the activity of detectors tuned to edges of different resolution and orientation. My strategy involved patterns which would be much more selective in the populations of detectors that were being stimulated. To eliminate the problem of common low frequency components, I used Gaussian modulated sinusoids or "Gabor functions". The spatial frequency content of Gabor functions is narrow and easily controlled by varying the period of the sinusoid. Luminance changes were eliminated by insuring that the time and space-averaged luminance of the Gabors were equal to that of the background.

Stimuli were displayed on a Hitachi high resolution monitor driven by a Grinnell graphics system. The viewing area was 14 by 12 degrees and had a mean luminance of 65 cd/m². When no stimuli were being displayed, the screen was uniform in luminance with the exception of a central cross-hair provided for fixation.

Targets were Gaussian modulated sinusoids, or "Gabor functions". These were created by calculating a sine-wave function, which varied around the mean luminance, and then multiplying the sinusoid by a circular Gaussian. The final product appeared as a circular patch of sine-wave about 1.7 degrees in diameter in which contrast was maximum at the center and decreased radially. Unless otherwise stated, phase of the sinusoid was 0 degrees with respect to the Gaussian function. This was necessary to insure that the space-averaged luminance of the Gabor would always be the same as that of the background. Contrast of the Gabor functions was determined by a matching procedure. The Gabor containing the highest central frequency, 10 c/deg, was set to 85% contrast. Physical contrast of all other Gabors was set to the same apparent contrast.

Experiments consisted of a series of trials in which the observer viewed a sequence of 4 frames. As shown in Figure 1, each frame contained four Gabors drawn on the circumference of an imaginary circle. Frames consisted of two pair of identical Gabors ("A" and "B"). In the experiments, A and B represented different values along the dimensions of spatial frequency or orientation. Figure 2 shows a picture of the actual display. In this case A and B differ in spatial frequency by 1.5 octaves. Frames 2 through 4 consisted of the same stimuli rotated by 45 degrees to new positions. Rotation changed only position, not orientation of the Gabors. The correspondence problem asks how the Gabors in frame 1 decide which Gabor in frame 2 is a proper match. Since the distance from A to B or another A is equal, there is no a priori reason for motion to be clockwise or counter-clockwise. If the difference between A and B can be used to determine correspondence, the A moves to A and B to B. Otherwise, direction should be ambiguous.

The observers' task in all experiments was to discriminate clockwise from counter-clockwise motion. On each trial, the sequence of 4 frames

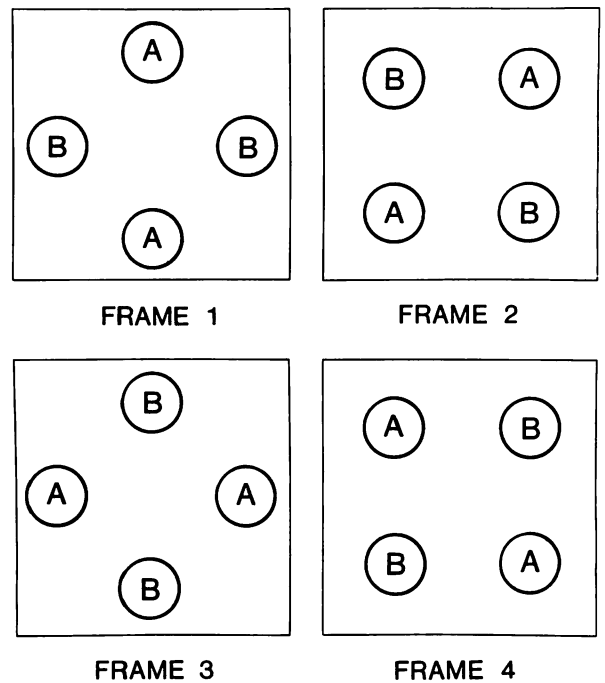


Figure 1

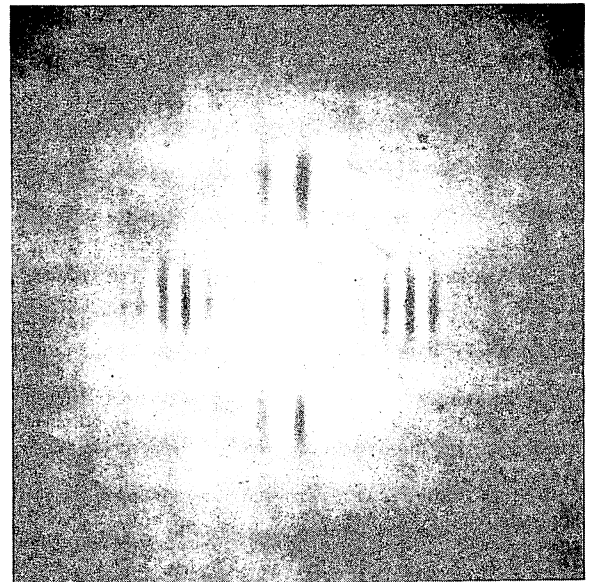


Figure 2

was shown twice in succession to produce rotation through 315 degrees. Frame duration was 84 msec (5 sweeps of the raster) and the interstimulus interval (ISI) between frames, during which only the uniform field was visible, was 17 or 50 msec. The only reason for choosing these time intervals was that they produced clear motion. The results reported below were robust and did not depend critically on any particular frame duration or ISI.

In the first set of experiments, A and B Gabors of different spatial frequencies. The left panel of Figure 3 shows the results obtained when A was fixed at 1.7 c/deg and the distance between

the centers of similar Gabors was 5.4 degrees. The actual distance between Gabors in successive frames was 2.3 degrees. This meant that there was no overlap in the position of a Gabor from one frame to the next. Ability to perceive direction of motion was at chance levels when A and B had the same value. As spatial frequency of B increased, discrimination between clockwise or counter-clockwise directions improved until performance was perfect. Observers reported that their ability to judge direction resulted from a coherent motion of the Gabors in a circular path. Data in the bottom panel show results obtained when spatial frequency of A was fixed at 5.0 c/deg. Clear spatial frequency tuning of the correspondence process is again evident for both observers. The tuning of the matching process is surprising sharp. I estimated that the curves fell to half-width/half-height in 0.5 to 1.0 octave, a value similar to that found for cells in the primary visual cortex and many other psychophysical experiments. I repeated the experiment with smaller diameter circles to produce different amounts of overlap between Gabor positions in successive frames. There was no evidence that matching was affected by whether or not overlap existed between successive frames (5).

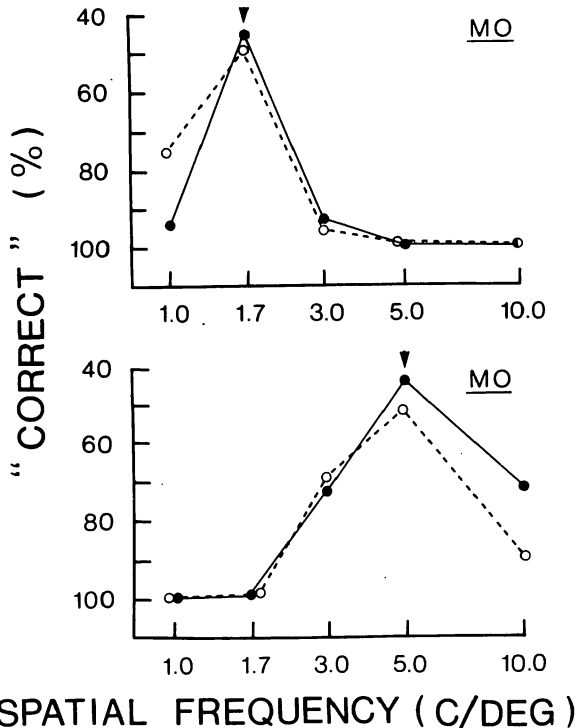


Figure 3

I earlier speculated that previous studies may have failed to find correspondence tokens because of the luminance flux which accompanied form presentation. To test this possibility, I repeated the basic experiment except that the background luminance was dark (actually 0.5 cd/m²) or half that of the Gabors (32.5 cd/m²). Observers failed to perceive strong coherent motion under any conditions.

I also investigated the possibility that orientation may be a token for correspondence. For

this experiment spatial frequency was set at 3.0 c/deg and orientation difference between A and B varied. As shown in Figure 4, correspondence exhibits a clear tuning for orientation. Differences of 22.5 degrees from the A value produced almost perfect performance. Although performance was excellent with orientation as the correspondence token, observers agreed that the coherence of the motion produced by orientation, although clear enough to make a correct judgment, was never as smooth as for spatial frequency.

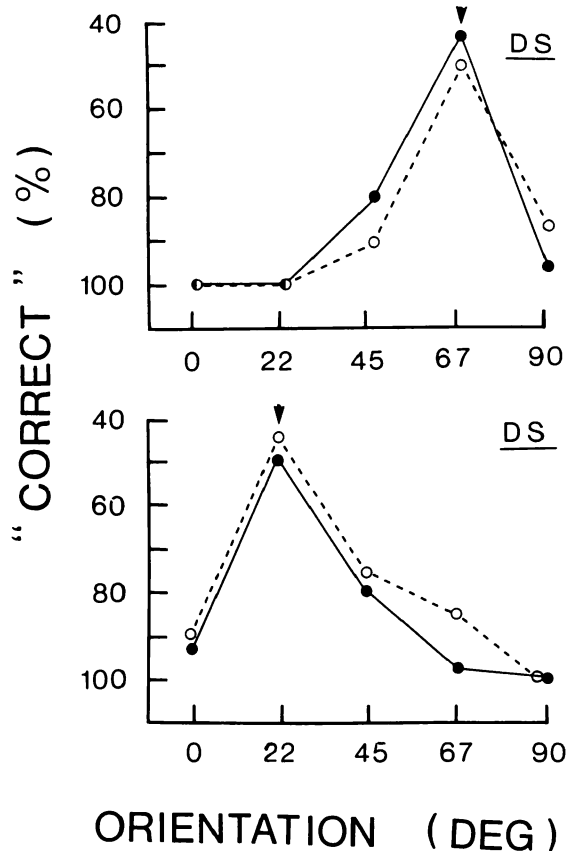


Figure 4

I concluded from these experiments that matching is based on similarity of spatial frequency and orientation. This contrasts greatly with the previous finding that form similarity seems to be important in apparent motion. My view is similar to that of Ullman (13), who has suggested that the many failures to uncover form tokens occurred because experimenters used relatively complex images. Shapes such as alphabetic characters consist of numerous tokens, so that any two images will usually have some tokens that match. By using simpler stimuli, isolated line segments, he found that orientation was an important token in matching. My conclusion differs slightly in that it is not the oriented lines that are important but rather the activity in populations of oriented, narrowband detectors. My analysis further differs from that of Ullman since I suggest that luminance transients play a large role in correspondence. The existence of transients may explain why orientation is not always found (3) to be a token, even with single

lines.

Some (2) believe that there are two mechanisms of motion correspondence, the "short-range" and "long-range" systems. Ullman also concluded that orientation was a token only when matching spanned very small spatial separations and stimulated short-range mechanisms. The step sizes used in my experiments were relatively large and likely activate long-range mechanisms. An additional demonstration shows that spatial frequency is also a token in short-range motion. Observers again viewed a series of 4 frames which were continuously recycled. Each frame contained either a 1) field of uniform luminance, 2) sine-wave grating, 3) square-wave grating or 4) missing fundamental-wave grating (MF) in which the sine is subtracted from the square. The MF wave looks much like the square-wave in that it contains an alternating series of sharp-edged bars. If frames 1 and 3 contain square-waves at 0 and 180 degrees and 2 and 4 are blank, then only flicker is perceived when the frames are cycled. If frame 3 contains a sine or square at 90 degrees and frame 4 a sine or square at 270 degrees, then the bars appear to march leftward. This presumably occurs because both sine- and square-waves contain the same fundamental which can be used to compute correspondence. If the wave in frames 2 and 4 is an MF, then only flicker is perceived, even though the patterns in all 4 frames appear very similar. This presumably occurs because the MF does not contain the low frequency component of the sine and square.

This demonstration dramatizes an important point: perception is based on processes occurring in a hierarchy of representations, but we "see" only the final product. The activity in spatial frequency and orientation selective detectors are used for correspondence matching but also provide the input to higher levels where feature and object descriptions are created. Our ultimate perception is that of moving objects containing features to which we have conscious access. This has led many experimenters to look for correspondence tokens in various feature or object domains. I believe this effort failed partly because correspondence is achieved at one level of representation while the features and objects were constructed at another. Too many experimenters employ images which indiscriminantly active detectors operating at low levels of representation. This is bound to obscure analysis of visual processing. The visual scientist's version of Occam's razor is that phenomena ought to be explained at the lowest possible level.

THE PROXIMITY HEURISTIC

The second unresolved question is whether matching employs two or three dimensional proximities. Initial studies (12) suggest that matching is based on 2-D proximities. In these studies, linear perspective was used to produce depth separation. I reexamined this conclusion using another depth cue, disparity (6).

The display was similar to that described above, except that each frame consisted of random dot stereograms, with A and B being disk-shaped submatrices of different disparity. Each disk was 1.3 degrees in diameter and lay on an imaginary

circle with a 1.8 degree diameter. Figure 5 shows how the display appeared to the observers. The pairs of disks seemed to float in front of the background at different depths. A small red square of 0 disparity provided at fixation point. Observers viewed a series of 8 such frames in which the disks' positions were rotated by 45 degree steps. If correspondence matching is based on 2-D proximity in the XY plane, then direction of rotation is ambiguous: frame 2 contains two possible matches equidistant from each object in frame 1. If 3-D proximity is used as the distance metric, then objects will appear to move to the neighbor in the same depth plane and therefore closer in 3-D space.

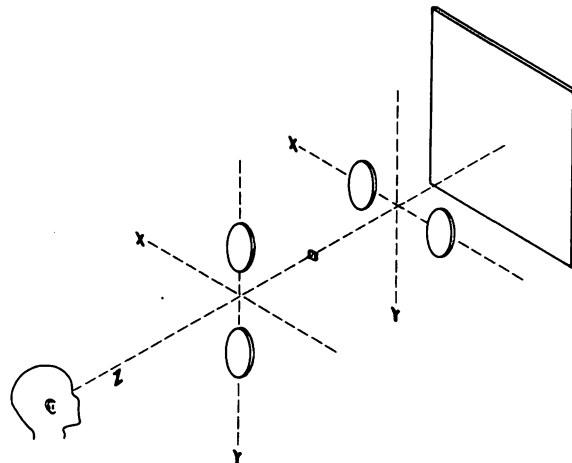


Figure 5

When viewing the sequence of frames, observers readily perceived clear rotational motion with disks moving to neighbors at the same depth. Figure 6 shows results from an experiment in which disparity was between pairs of disks was varied symmetrically around 0 disparity. Each data point represents the percentage of times direction of motion was toward the neighbor at the same apparent depth. For large disparities, almost all judgments were consistent with the 3-D interpretation. At a disparity of 0, since there were two equidistant neighbors, direction was ambiguous, and fell to chance. I repeated some of our observations with displays where the radius of the circle was larger (2.5 degrees) and smaller (1.2 degrees). Similar results were obtained.

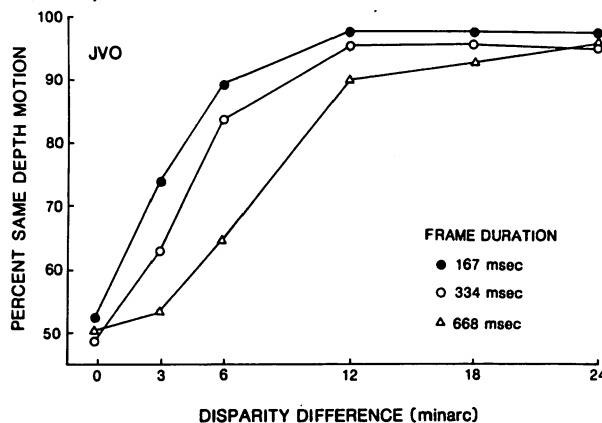


Figure 6

In the experiment described above, images had no monocular cues. We also were forced to use long durations since the disks would dissolve into the background if the motion were too fast. We repeated the experiment with the light squares of the disks darkened so that the observers saw gray blobs floating against the background. This produced monocular cues and allowed us to use a higher frame rate, 84 msec per frame. The new display also produced a compelling circular motion with disks moving toward neighbors in the same depth plane, and observers were almost 100% correct in discriminating the motion path. To be sure that the monocular cues did not contribute to ability to judge direction, observers attempted the task with one eye occluded. Results showed the observers performing at chance.

Our results are consistent with the view that correspondence matching utilizes 3-D proximities. In a subsequent experiment, we further demonstrated that distance in the X, Y and Z planes can be traded off so that objects will appear to move in depth when the nearest neighbor lies at a different disparity. Based on our evidence, it might be expected that correspondence matching by computer would be enhanced by assignment of depth/disparity to images in each frame. This is confirmed by Jenkin (7), which found that disparity information was effective in producing more accurate matching.

COMPUTING CORRESPONDENCE

To compute correspondence both a representation and an algorithm must be specified. A possible representation is suggested by the studies described above. Images would be represented by the activity of a multidimensional detector array, where each dimension represents a "primitive continuum". These are dimensions on which detectors are selective, responding only to a narrow range of values. Detectors may be continuously mapped or the dimension can be resolved into discrete segments to facilitate use of algorithms such as a Hough transform. If the dimensions are resolved, degree of resolution might be suggested by psychophysical studies. For example, the first experiments indicate that spatial frequency could quantized into 0.5-1.0 octave steps.

A metaphor for this array is an n-dimensional space where each detector occupies a single location. The detectors are blobs, rather than points, since detectors have a non-zero bandwidth in most dimensions. Important dimensions, suggested by the studies described above, include spatial frequency, orientation, X, Y and Z coordinates (or possibly disparity, depending on whether other depth cues are employed). Sustained and transient detectors would have to be separately represented. There may be other important dimensions as well. For example, I have been examining whether color might be added to this list, but the results so far have been ambiguous.

To perform correspondence matching the activity of points at time t_1 is compared to

activity of points at t_2 . A correspondence algorithm might try to match up similarly tuned detectors, i.e., nearest neighbors in the detector space. The preference metric then becomes an index of proximity in the space and would be calculated by summing the (weighted?) proximities in each of the n dimensions. To accomplish this, the metric of the detector space must be found. In the simplest case, dimensions are independent and the metric is "city-block". Proximity is then simply a sum of the distances in each dimension. However, it is more likely that the dimensions are not independent so that computing proximity would not be so simple. For example, if the metric were Euclidean, then distance would be derived from taking the square-root of the sum of squares in each dimension. The situation could be even more complicated if different planes have different Minkowski metrics. Once the spatial metric is known, then matching can proceed by any of the standard algorithms, such as relaxation labeling.

However, given the detector space representation, there are many possible variations to the scheme outlined above. For example, some (10) suggest that each resolution channel be computed independently while others (11a) believe that cross-channel correspondences should be determined first. However the correspondence is computed, it apparently must consider a low level representation rather than one in feature or object domains.

REFERENCES

- (1) Blakemore, C. and F. Campbell (1969) On the existence of neurons in the human visual system sensitive to the size and orientation of retinal images. *J. Physiol.*, 203, p. 237.
- (2) Braddick, O. (1974) A short-range process in apparent motion. *Vis. Res.*, 14, p. 519.
- (3) Burt, P. and G. Sperling (1981) Time, distance and feature trade-offs in visual apparent motion. *Psych. Rev.*, 88, p. 171.
- (4) Green, M. (1984) Masking by light and the sustained-transient dichotomy. *Percept. & Psychophys.*, 35, p. 519.
- (5) Green, M. (1986) What determines correspondence in apparent motion? *Vis. Res.*, in press.
- (6) Green, M. and J. Odom (1986) Correspondence matching in apparent motion: Evidence for 3-D internal representation. *To be published.*
- (7) Jenkin, M. (1983) Tracking three dimensional moving light displays. *Proc. ACM Interdisc. Work. Mot.*, p. 66.
- (8) Kolers, P. and J. Pomerantz (1971) Figural changes in apparent motion. *J. Exp. Psych.*, 87, p. 99.
- (9) Marr, D. (1982) *Vision.*
- (10) Marr, D. and E. Hildreth (1979) The theory of edge detection. Tech. Report from M. I. T.
- (11a) Mayhew, J. and J. Frisby (1981) Psycho-physical and computational studies towards a theory of human stereopsis. *Artif. Intell.*, 17, p. 349.
- (11) Navon, D. (1976) Irrelevance of figural identity for resolving ambiguities in apparent motion. *J. Exp. Psych. HP & P*, 2, 130.
- (12) Ullman, S. (1978) Two dimensionality of the correspondence process in apparent motion.

Perception, 7, p. 683.

(13) Ullman, S. (1980) The effect of similarity between line segments on the correspondence strength of apparent motion. Perception, 9, p. 617.

(14) Wertheimer, M. (1912) Experimentelle Studien uber das Sehen von Bewegung. Z. Psychol., 61, p. 161.

THREE PROCESSING CHARACTERISTICS OF TEXTURE DISCRIMINATION

Terry M. Caelli

Department of Psychology, The University of Alberta

Edmonton, Alberta T6G 2E9

Abstract

In this paper we examine the idea that texture segmentation comes about by the differential outputs of detectors (non-linear associative filters) computed at each resolvable position on the textured surface. Further, we consider some of the conditions under which "primary" detector outputs are dynamically compared and associated to develop into a smaller set of "texton" profiles which capture the predominant differentiating features of the texture regions. Comparisons to human psychophysical results are made.

KEY WORDS: texture segmentation, associative networks, orientation detectors, adaptability

1. Introduction.

For a biological visual system endowed with a multitude of cells which apparently act as feature extractors or filters, it seems reasonable to presume that visual texture segmentation may come about by the differential responses of such detectors over the textured region. This proposal has received experimental and mathematical attention over the past decade with one-dimensional grey-scaled textures (Richards, 1979; Harvey & Gervais, 1978) and two-dimensional textures (Caelli & Julesz, 1978; Caelli, 1982, 1985). However, only until recently has a full computational model been proposed which produces segmentation as a function of such "texton" (Julesz, 1981) outputs, and this paper extends the above analyses in a number of ways (Caelli, 1985).

Here texture segmentation is viewed as having three component processes: (1) spatial decomposition, (2) dynamical associative processing, and (3), classification of textured regions. The specific aims of this model are to enable segmentation when the textures consist of sparse micropatterns; to create networks which will extract, or

adapt to, the predominant features of the texture; and to use a classification procedure which is adaptive to the outputs of such detectors.

2. The Model.

2.1 Level I processing: Spatial decomposition and activity profiles.

The initial process of texture segmentation is envisaged to involve the registration of the input (foveal) texture through the parallel outputs of many detectors whose responses are determined by some non-linear transformation of their cross correlation with the input. Assuming a relatively fixed "retinal pre-processor," having opponent center-surround receptive fields, the primary information to be processed must have differential, or band-pass, components emphasized. Further to this, we assume the existence of a relatively fixed primary projection area where such image derivative information is further classified (encoded) by cortical edge and bar detectors whose outputs are a non-linear function of the cross-correlation of the detector's profile with the input image. That is, we assume that the response $R_i(x,y)$ of a detector d_i at retinotopic position (x,y) is determined by:

$$\sum_{\alpha, \beta} d_i(\alpha, \beta) = \text{const.} \quad (1)$$

$$\text{and } R_i(x,y) = \text{const} + Y\psi\{d_i o I\}, Y = \text{constant.} \quad (2)$$

o denoting cross-correlation between the detector and image (I)

$$d_i o I(x,y) = \sum_{\alpha, \beta} d_i(\alpha, \beta) I(x+\alpha, y+\beta), \quad (3)$$

and

$$-1 \leq \psi_\delta(z) = \frac{1-e^{-\delta z}}{1+e^{-\delta z}} \leq +1, \delta = \text{constant.} \quad (4)$$

In our simulations we have used

$$R_i(x,y) = 128 + 127 \psi_\delta\{d_i o I\}, \delta = 0.03, \quad (5)$$

to fit in with an 8-bit response range. The non-linear transducer enables one to move smoothly from square wave ("ideal" edge and bar) to gaussian modulated sinusoid (Daugman, 1983) representations for edge and bar, or orientation detectors, via δ in (4). Orientations and sizes of the detectors were chosen to fit with a large number of experimental results on human texture discrimination showing the inability of observers to resolve image orientations to better than $\pm 5^\circ$ (Caelli, 1982; Beck, 1983). With evidence that such receptive, or "perceptive", fields are limited in size to $\pm 1/8$ octave, or 1 1/2 cycles to 1/e decay of a gaussian aperture, we have generated 24 fundamental orientation detectors over 7x7 pixel kernels (relative to 128x128 pixel textures) satisfying these profile constraints for both edge (odd) and bar (even) detectors.

We secondly assume that the response profile for each detector is 'rectified' into an "activity" profile.

$$A_i(x,y) = |R_i(x,y) - \text{const}|, \text{const} = 128. \quad (6)$$

2.2 Level II processing: Adaptive control and selection of critical features.

In contrast to representing texture codes by detectors defined at different size scales, in gaussian pyramids, etc. (see Burt & Adelson, 1983), which would be capable of responding to texture regions in areas greater than the actual micropattern size--the approach adopted here remains at the resolution of the basic texture--though this is not a necessary restriction. Further, the process of texture region "filling-in" (impliment) is seen as a dynamic process involving the iterative activity of activated detectors in terms of how their responses may spread over contiguous regions in a summative (averaging) fashion. This is analogous to relaxation in image processing (Hummel & Zucker, 1983) where the strength of a given spatial response is reinforced or inhibited as a function of neighbouring collaborative or opposite evidence. In particular, it is assumed that the activity of a given detector d_i determined by (6) is updated dynamically by the following (associative) "texture processing equation":

$$A_i^{t+1}(x,y) = \frac{1}{\alpha\beta} \sum_{\alpha\beta} A_i^t(x+\alpha, y+\beta) + \sum_{j=1}^n w_{ij}^t A_j^t(x,y) \quad (7)$$

where (α, β) corresponds to the "region of influence" at each iteration. w_{ij}^t is the coupling, or associativity, between two activity profiles which can either be fixed or adaptive. For the fixed case we have used the well-known "mass-action" formulation for detector

coupling (Grossberg, 1982), where we have set:

$$w_{ij}^t = \begin{cases} k \dots (i,j) \text{ being (edge, bar) pairs at the same orientation} \\ -\left(\frac{1}{n-2}\right)^k \dots \text{otherwise.} \end{cases} \quad (8)$$

for n detectors and k being the coupling strength such that

$$\forall i, \sum_{j=1}^n w_{ij}^t = 0. \quad (9)$$

In general, it seems unlikely that the (neural) connectivity between such detector planes can be defined by a single stationary matrix w_{ij} over space and time. Like Hebb (1949) and Fukushima (1984), we assume that the process of perceptual learning and adaptation involves the dynamic updating of w_{ij} as a function of the detector's response strengths and correlations. For these reasons, our interests are also focused upon investigating formulations for w_{ij}^t such as:

$$w_{ij}^t = r_{ij}^p \{a_{ij} A_j^t(x,y) + b_{ij}\} \quad (10)$$

where a_{ij} and b_{ij} correspond to slope and intercept regression coefficients of A_j on A_i , p to the degree to which this correlated information is combined with the response of A_i to result in "new" detector profiles. This dynamical system converges to strong "attractor" detectors which (as will be shown) have receptive fields related to the first few eigenvalues of the coupling matrix. Using (10) in (7) requires normalization as:

$$z_i^{t+1}(x,y) = \frac{1}{[1 + \sum_{j=1}^n r_{ij}^p]} \left[\frac{1}{\alpha\beta} \sum_{\alpha\beta} z_i^t(x+\alpha, y+\beta) + \sum_{j=1}^n r_{ij}^p [a_{ij} z_j^t(x,y) + b_{ij}] \right] \quad (11)$$

for $0 \leq p$ and even. The first component:

$$\frac{1}{\alpha\beta} \sum_{\alpha\beta} z_i^t(x+\alpha, y+\beta)$$

being an averaging process (moving spatial window), is clearly "local" and restricted to a given detector plane. That is, activity within a given plane spreads as a function of the neighbouring activity of the same detector type and converges to a mean level of activity. Secondly, this activity is reinforced as a function of the degree to which other detectors exhibit similar graded responses over the full texture regions--a global cooperative component--represented by the last component in (11): "synergesis".

This has the effect of combining correlated detector responses and converging to common ("attractor")

profiles, so reducing the number of different detectors. Again, it should be noted that the solutions are critically dependent on the input signal. In the case of texture segmentation, where broad spatial regions have to be "labeled", inhibitory forms of W_{ij} seem inappropriate as they differentiate the detector responses in further ways. This, in turn, would not produce the percept of contiguous spatial regions, and would be more useful in pattern recognition where it is precisely these differentiated dimensions which are needed.

Finally, we consider the network to "complete" its activity when it reaches near equilibrium state; as measured by:

$$\frac{1}{n\alpha\beta} \sum_{\alpha,\beta} \left[\frac{A_i^{t+1}(x,y) - A_i^t(x,y)}{A_i^t(x,y)} \right] \leq \delta, \quad (12)$$

δ being near zero (in our case $\delta=0.02$). Here n corresponds to the number of detectors and (α,β) to the image size.

It should be noted that formulation (11) is an example of an associative network whose coupling undergoes adaptation, and, if we consider the problem of texture segmentation as primarily involving the extraction of the main dimensions for segmentation, then it is the eigenvalues of W_{ij} which are critical. Further, we could also claim, like Kohonen (1977) and Anderson, Silverstein, Ritz and Jones (1977) that W_{ij}^t --the network associativity at time t is the primary attribute of the model rather than the detector states, per se. However, the author feels these claims to be too strong since both $\{A_i\}$ and $\{W_{ij}\}$ are mutually dependent. However, in the simulations to be reported we shall observe the behaviour of the eigenvalues of W_{ij} to investigate how these adaptive processes are changing the dimensionality of the problem.

2.3 Level III processing: Region classification and decision criteria.

Since textured regions are proposed to appear as a function of position response differences in "feature space", the appropriate classification process seems to be the minimum distance classifier (MDC, Ahmed & Roa, 1975). This method determines whether a pixel falls into one of two textured regions as a function of whether it is closer to the centroid of the texture or not. The MDC determines the discriminant (function) hyperplane which constitutes the locus of points equidistant between both centroids, and of the form:

$$g(a_1, \dots, a_n) = \sum_{j=1}^n [\bar{a}_{1j} - \bar{a}_{2j}] a_j - \frac{1}{2} \left[\sum_{j=1}^n \bar{a}_{1j}^2 - \sum_{j=1}^n \bar{a}_{2j}^2 \right] \quad (13)$$

where $(a_1 \dots a_n)$ correspond to the feature dimensions--in this case detector outputs. \bar{a}_{ij} corresponds to the mean value for group (texture) i on feature j , while a_j corresponds to a given input texture pixel feature weights. The pixel is classified as a function of the sign of $g(a_1 \dots a_n)$.

To introduce a degree of "fuzziness", or "segmentation strength", into this procedure, it would be adequate to use the distance between the means over the standard deviation of both sets (or average t statistic):

$$\bar{t} = \frac{1}{n} \sum_{i=1}^n t_i, \text{ for } t_i = \frac{\bar{a}_{1i} - \bar{a}_{2i}}{\sqrt{\frac{s_{1i}^2}{n_1} + \frac{s_{2i}^2}{n_2}}} \quad (14)$$

where n_1, n_2 correspond to the number of pixels in each textured region, s_{ji}^2 to the appropriate variance statistic.

This function not only indicates that adding common features to the textures would decrease perceptual segmentation, but would also decrease if more variability in detector outputs was observed over either, or both, regions.

3. Simulations and Conclusions.

We first summarize the main properties of the model:

- (T₁) Detector activity is determined by the rectified response profiles as a result of detector cross-correlation with the incoming texture, according to (1)-(7).
- (T₂) The activity of a given detector at time t and position (x,y) is determined by the degree to which neighbouring regions are also active with respect to this detector and the activity of others.
- (T₃) The associativity between detector arrays (i,j) is adaptive to their responses.
- (T₄) A perceptual classification is made after this system of dynamically responding detectors reaches equilibrium.
- (T₅) Classification of positional information into textured regions is accomplished by a weighted form of the minimum distance classifier, weighted by the total texture "entropy".

We have implemented all three proposed processes (convolution, implem/cooperativity, and classification) to quantitatively observe the behaviour of the system with four critical texture pairs consisting of grey-scaled textures differing in

granularity, simple textures differing in orientation, and those differing in micropattern space characteristics: T,L, etc. We have chosen these latter two pairs since it has been (a) shown that they differ in discriminability, and (b) it has been proposed that they require "end-of-line" and intersection detectors to discriminate (Julesz, 1984)--the latter we can disprove. These are shown in Figure 1 together with the outputs of the classification procedure, using (13) to represent the relative "strength" of discrimination: Convergence usually occurred within 5-7 iterations.

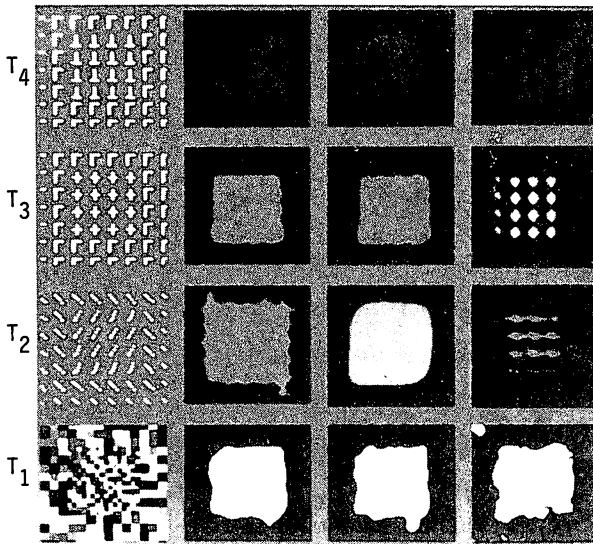


Figure 1.

Input textures (left column) and segmentation resulting from the outputs of 24 detectors with no associativities (second column), associativities as defined by (11) (column three) and inhibitory mass action (column four, equation 9): $K=0.05$ in (8). Contrast reflects segmentation strength according to equation (14).

To illustrate the effects of associativities on decreasing the dimensionability of the classification process, Figure 2 shows the eigenvalues for the solutions shown in columns two and three of Figure 1. Such reductions in "dimensionality" are clearly related to the iterative process converging to common strongly active detector profiles and inhibiting less active and isolated ones.

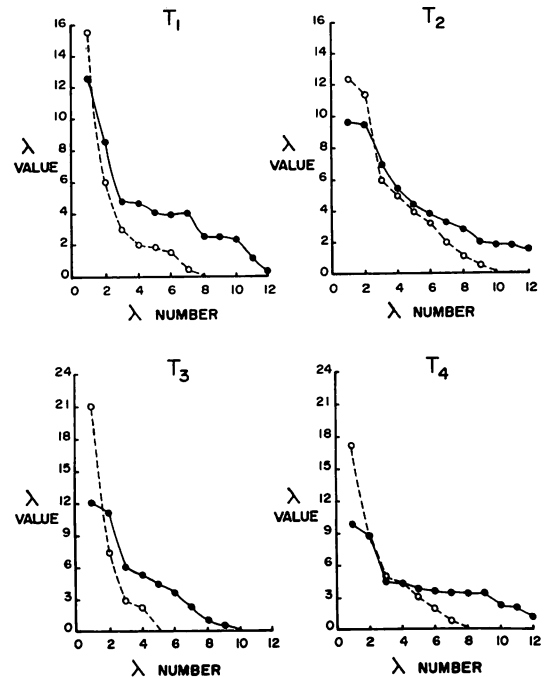


Figure 2.

Eigenvalues for the non-associative (solid lines: column 2, Figure 1) and associative (dashed lines: column 3, Figure 1) segmentation processes. T_1 to T_4 correspond to the 4 textures shown up Column 1 of Figure 1.

What connects the texture processing equation (7) and the "texton" approach is that such cooperative networks decrease the dimensionality of the problem to the more strongly active--though "adaptively generated"--detectors or dimensions. That is, the profile of each detector in the process described by (8) is not stationary but, rather, is adapted by the energy it is designed to process and the activity of other units. Indeed, the actual profile at any time is recoverable by inverse filtering.

This model for texture segmentation is algebraically similar to a class of models for pattern recognition based on the associative (coupled) activities of large numbers of computational units whose activity profiles adapt to the signal and network states (see Kohonen, 1977; Fukushima, 1984). The main difference lies in how each computational component is interpreted, and the involvement of a classification scheme at the end, which actually produces the textured regions. In this sense the model is not formally dependent upon the initial edge and bar

detectors chosen, but rather on the ways in which their outputs are correlated over space and time according to (7).

In the present texture processing model the nature of the decomposition, and so dimensions, of a given texture segmentation task is dependent on the signal and the type of coupling operating between the computational units. If the visual system (or, indeed, the scientist) were to choose detectors which satisfied absolute orthogonality ($d_i \cdot d_j = 0$, * being convolution) then, from a mathematical perspective the ideal detector conditions would be present and the cooperative processes defined by (11) and (12) would not be required. However, the Implementation process would be involved, along with the classification algorithm. However, one assumption here is that the visual system is not that precise in creating detectors which, a priori, are so independent. Rather, the idea is that the visual system converges on the central detector profiles by adaptation processes like those described here--being signal dependent and network specific.

In conclusion, then, we have extended an earlier model for texture segmentation initially related to the "heuristics" of Julesz and Bergen (1983) for "preattentive" and "attentive" visual processes in spatial vision. The model has three components: decomposition (via cross-correlation), local and global processing, and classification. Though many questions still remain unanswered, our results suggest that these mechanisms, in a psychophysical sense, represent the types of processing involved in texture segmentation. The main result here is that the enumeration of detector profiles is but one part of the texture discrimination process and that the detector profiles "attended to" by the visual system are signal dependent and not fixed and invariant over all texture types, but also resultant from the underlying cooperative processes which generate "texton" classes to optimize the classification process by as few dimensions as possible. The proposed model does not solve the apparent rotation invariance processing characteristics of texture micropatterns, nor does it propose only one form of cooperative process. In this case we have found that the global process defined by (7) is efficient in reducing the dimensionality of the classification problem and have proposed that such coupling cannot be inhibitory if some form of "filling-in" is required.

References

- Ahmed, N. & Rao, K. (1975). Orthogonal Transforms for Digital Signal Processing. Berlin: Springer.
- Anderson, J.A., Silverstein, J.W., Ritz, S.A. & Jones, R.S. (1977). Distinctive features, categorical perception, and probability learning: some applications of a neural model. Psychological Review, 85, 413-451.
- Burt, P.J. & Adelson, E.H. (1983). The Laplacian pyramid as a compact image code. IEEE Transactions on Communications, COM-31(4), 532-540.
- Beck, J. (1983). A theory of textural segmentation. In Jack Beck (Ed.) Human and Machine Vision. New York: Academic Press, 1-38.
- Caelli, T. (1982). On discriminating visual textures and images. Perception and Psychophysics, 31, 149-159.
- Caelli, T. (1985). Three processing characteristics of visual texture segmentation. Spatial Vision, 1(1), 19-30.
- Caelli, T. & Julesz, B. (1978). On perceptual analyzers underlying visual texture discrimination: Part 1. Biological Cybernetics, 28, 167-175.
- Daugman, J. (1983). Six formal properties of two-dimensional anisotropic visual filters: Structural principles and frequency/orientation selectivity. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13, 882-888.
- Fukushima, K. (1984). A hierarchical neural network model for associative memory. Biological Cybernetics, 50, 105-113.
- Grossberg, S. (1982). Studies of the Mind and Brain: Neural Principles of Learning, Perception, Development, Cognition and Motor Control. Boston: Reidel.
- Harvey, L.O. & Gervais, M.J. (1978). Visual texture perception and Fourier analysis. Perception & Psychophysics, 24, 534-542.
- Hebb, D.O. (1949). The Organization of Behavior. New York: John Wiley.
- Hummel, R. & Zucker, S. (1983). On the foundations of relaxation labelling processes. IEEE: PAMI, 3, 267-287.
- Julesz, B. (1981). Textons, the elements of texture perception and their interactions. Nature, 290, 91-97.
- Julesz, B. (1984). Toward an axiomatic theory of preattentive vision. In Dynamic Aspects of Neocortical Function, G. Edelman, W. Gall, W.M. Cowan (Editors).
- Julesz, B. & Bergen, J. (1983). Textons, the fundamental elements in preattentive vision and perception of textures. Bell Syst. Tech. Journal, 62, 1619-1645.

- Kohonen, T. (1977). Associative memory--a system-theoretic approach. New York: Springer.
- Richards, W. (1979). Quantifying sensory channels: Generalizing colorimetry to orientation and texture, touch and tones. Sensory Processes, 3, 207-229.

PARALLEL ARCHITECTURES FOR MACHINE VISION

Steven L. Tanimoto

Dept. of Computer Science
University of Washington

ABSTRACT

The thrust toward parallel processing in machine vision has been unusually intense for several reasons: there is a tremendous amount of parallelism inherent in algorithms for image processing; the spatial regularity of image data structures lends itself well to VLSI implementations; and programming for parallel systems is probably better understood in the context of image processing than in any other realm of application. The SIMD variety of parallel computer systems matches well with many machine-vision problems, and these systems permit large amounts of parallelism to be applied to a problem with relatively little programming effort and relatively high efficiency.

symbolic operations quite well, but they are not efficient in transforming information from iconic form or vice-versa. Some of the proposals for bridging this gap are discussed.

KEYWORDS: machine vision, image processing, parallel processing, architecture, pyramid machine.

Parallel architectures for machine vision may be classified according to the dimension across which they achieve their parallelism. Most common are those that achieve parallelism across an image; their different processing elements treat different parts of the image simultaneously. Other architectures are pipelined, performing at any one time the different steps of an algorithm on part of a stream of pixels. The image-parallel and the pipelined architectures do not exhaust the list, for there are more dimensions for parallelization: across goals, across pixels, and across neighborhoods, to mention several.

Some parallel architectures can take advantage of more than one dimension of parallelism. The "pyramid-machine": architecture is one of these. This architecture combines features of meshes such as the "CLIP4" and the "Massively Parallel Processor" with tree machine capabilities, yielding a system with considerable flexibility and efficiency. Some features of a pyramid machine under study at the University of Washington are described. Algorithms for pyramid machines have been investigated by several research groups, and they can be classified according to the control and data-flow paradigms they follow.

A challenging problem exists today in developing architectures that can bridge the "iconic-to-symbolic gap". The high-performance architectures for vision that currently exist focus their efforts on either the pixel-level operations or

DETERMINING DISPLACEMENT FIELDS
ALONG CONTOURS FROM IMAGE SEQUENCES

Patrick BOUTHEMY

IRISA / INRIA
Campus de Beaulieu
35042 RENNES, France

ABSTRACT

We propose a framework for image flow analysis consisting of three major stages; i.e.:

- determine moving edges by some local process;
- integrate motion information along linked contours;
- propagate motion estimation through homogeneous regions obtained inside these contours.

This paper is concerned with the two first stages. A procedure, based on some local modeling and maximum likelihood scheme, has been designed to perform the first step. After some linking process, constraints provided by the measurements gained from the first stage can be combined to compute the velocity field along contours, by minimizing some simple functional. To this end, a gradient algorithm is used with a recursive estimation from one point to its successor in the chain.

RESUME

Nous proposons un schéma d'obtention du champ des vitesses dans une séquence d'images s'articulant en trois étapes, à savoir :

- déterminer localement les éléments de contour en mouvement;
- intégrer l'information de mouvement le long des lignes contours chaînées;
- propager l'estimation du mouvement à l'intérieur des zones homogènes délimitées par ces lignes contours.

Le papier traite des deux premières étapes. Une procédure, basée sur une modélisation locale et un critère de maximum de vraisemblance, a été conçue afin de réaliser le premier point. Après chaînage, les mesures issues du premier niveau peuvent être combinées afin de calculer le champ des vitesses complet le long des lignes contours via la minimisation d'une fonctionnelle simple. A cette fin, un algorithme de gradient est mis en oeuvre avec une récurrence de point en point le long de la chaîne contour.

KEYWORDS: image sequence, moving edge determination, motion estimation, local modeling, maximum likelihood test, stochastic gradient.

I INTRODUCTION

Image sequence analysis has received more and more attention since 70's. In particular substantial studies have been concerned with motion estimation across changing two-dimensional images. Two main motivations have subtended there research efforts. First, motion computation represents an attractive challenge in order to design some robust, tractable and general-purpose method. On the other hand, application areas never stop broadening, [1].

Meteorological applications (determining wind fields owing to cloud motion estimation, [2]), military domain (target tracking) were among pioneer ones. Then came interframe image coding for broadcast television or videoconferencing purpose [3,4]. For a few years, other potential applications have appeared: biomedical (e.g., angiocardiology [5]), robotics (mobile robot, [6]), traffic monitoring, graphics ... These new domains are not only interested in two-dimensional motion as it is, but as intrinsic features conveying information about the depicted 3D-scene. Indeed motion in the imaging plane provides primary cues to relative depth, structure and 3D-movements of objects in space [7,8].

The motion in the imaging plane is usually referred to as the "optic flow". Optic flow can be represented as a vector field: the field of apparent velocities of brightness patterns in the image due to relative motion of camera and objects in space. (As one's uses a discrete representation of an image sequence, displacement vector fields and velocity vector fields are usually confused, although mathematically of different nature).

Discriminating discontinuities in the velocity field is a key problem in motion estimation schemes whatever they are. Indeed, feature-based methods require cooperative matching procedures [9], and gradient-based methods involve some smoothing constraint [10,11]. Thus, we have designed a method whose first task is to cope with these discontinuities, which are tied to contours in the image, such as occluding contours, joint ones ...

We propose a framework for image flow analysis consisting of three major stages; i.e.:
determine moving edges by some local process;
link these edges and integrate motion information along contours;
propagate motion estimation through homogeneous re-

gions obtained inside these contours.

The two first issues are addressed in this paper.

The first step can be considered as an early processing whose output contains location and spatial direction of an edge element and component of its displacement in the direction perpendicular to the local orientation of the edge. It is well-known that only such partial motion information can be reached by local operations (this point is often referred to as the aperture problem). A maximum likelihood method, based on some local modeling has been designed for this purpose.

Then constraints provided by these measurements gained from the first stage can be combined to compute the velocity field along contours, if however variations in spatial orientations occur along such contours. This computation results from the minimization of some simple functional by a stochastic gradient algorithm.

II LOCAL DETERMINATION OF A MOVING EDGE

II.1 - Modeling of a moving edge

An image sequence is considered as a 3D-space (x, y, t) . A spatial 2D-edge in an image is modeled as a small local linear segment. Hence a moving 2D-edge is locally modeled as a small planar patch in the spatio-temporal 3D-space (x, y, t) . The direction θ (w.r.t. the x-axis) of the 2D-edge centered in (x_0, y_0) in the xy-plane at time t_0 and its velocity $\vec{V} = (\frac{dx}{dt}, \frac{dy}{dt}, 1)$ determine the orientation of this planar patch (see figure 1). This planar modeling is equivalent to the first order approximation that most gradient-based methods take into account.

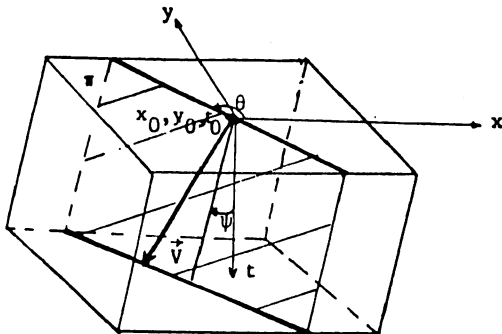


Figure 1: Local modeling of a moving edge as a planar patch.
 $\theta \in [0, \pi[$; $\psi \in [0, \pi/2[$

Let us consider an elementary volume Π , in the 3D-space (x, y, t) , located around point $\ell = (x_0, y_0, t_0)$. Two hypotheses (or local configurations) can be acting:

H_0 : there is no spatio-temporal edge inside Π ; then the intensity distribution within Π is modeled as $c_0 + b$, where c_0 is a constant and b denotes a zero-mean Gaussian noise with variance σ^2 .

H_1 : there is a spatio-temporal edge inside Π , i.e. a small planar patch P splitting Π between two sub-regions, Π_1 and Π_2 . Then the intensity distribution is modeled as: $c_1 + b$ within Π_1 ; $c_2 + b$

within Π_2 , where c_1 and c_2 are two different constants.

The orientation of the planar patch can be defined by the two following angles: θ (w.r.t. to the x-axis) and ψ (w.r.t. to the t-axis) as illustrated in Figure 1. The component V^\perp of \vec{V} perpendicular to the spatial 2D-edge and projected in plane $t=t_0$, is given by: $V^\perp = \tan \psi$. It is obvious that only this component V^\perp can be inferred from the local determination of this planar patch. Note that the case of a static edge belongs to hypothesis H_1 ; $\vec{V} = (0, 0, 1)$ and $\psi = 0$. Indeed such an edge will be considered as a "moving" edge, whose displacement is zero.

The problem now is how to select one hypothesis versus the other one. The test in order to decide between these two hypotheses will be designed using some maximum likelihood scheme.

II.2 - Maximum likelihood test

Details in mathematical developments can be found in [12], concerning the maximum likelihood test designed for detecting moving edges along with estimating their parameters. It is expressed by:

$$\max_{\ell, \theta, \psi} \max_{c_1, c_2} \min_{c_0} \text{LRV} \geq \lambda \quad (1)$$

where LRV is the log-ratio of likelihood functions L_1 and L_0 , respectively associated with hypotheses H_1 and H_0 . The likelihood function is merely the joint probability density function of the intensities within elementary volume Π . It is easily derived as Gaussian distributions are involved and independent intensity random variables are assumed. λ is a predetermined threshold.

Clearly, hypothesis H_1 is selected if the obtained maximum value of LRV, is greater than λ . Then one can conclude that a moving edge is located at ℓ with spatial direction θ and "perpendicular" velocity $V^\perp = \tan \psi$, where ℓ, θ, ψ are precisely values of (ℓ, θ, ψ) which have satisfied the mentioned criterion (1).

Yet one problem arises. No analytical closed-forms can be derived to express the optimal estimators θ, ψ corresponding to the geometrical characteristics of the model. Thus a predefined set of given configurations $\phi_j, j=1, \dots, G$ will be considered

For a given geometric configuration (θ_j, ψ_j) , the optimal estimators $\hat{c}_i (i=0, 1, 2)$ concerning intensity aspects satisfy:

$$\frac{\partial \text{LRV}(\ell, \phi_j)}{\partial c_i} = 0$$

which leads to

$$\hat{c}_0 = \frac{1}{n} \sum_{p \in \Pi} f(p); \quad \hat{c}_1 = \frac{1}{n_1} \sum_{p \in \Pi_1} f(p); \quad \hat{c}_2 = \frac{1}{n_2} \sum_{p \in \Pi_2} f(p) \quad (2)$$

where $f(p)$ are observed intensity values within Π , n (resp. n_1, n_2) is the number of points within Π ; (resp. Π_1, Π_2).

II.3 - Computational scheme

It turns out, [12], that maximizing LRV comes to maximizing the following expression:

$$CRV(\ell, \phi_j) = \sqrt{\frac{n_1 n_2}{2n}} |\hat{c}_1 - \hat{c}_2| \quad (3)$$

Using (2) and (3), we can write function $CRV(\ell, \phi_j)$ in the form:

$$CRV(\ell, \phi_j) = \left| \sum_{m \in M} a_j(m) f(\ell+m) \right| \quad (4)$$

where M is a set of vectorial indices such that $\{\ell+m, m \in M\}$ represent all points of volume Π , and coefficients a_j 's only depend on a predefined configuration ϕ_j . Indeed, the computational implementation of this local estimation process merely consists of convolution operations.

The process for determining moving edges between two successive images can be summarized as follows. For each point ℓ in the first image:

- . calculate $LRV(\ell, \phi_j)$ for each mask $\{a_j\}$ corresponding to geometric configuration ϕ_j ;
- . select orientation ϕ_k which maximizes function LRV ; if $LRV(\ell, \phi_k) \geq \lambda$, then a moving edge is said to be potentially present at point ℓ , whose estimated parameters are ϕ_k and associated likelihood value denoted by $CONF(\ell) = LRV(\ell, \phi_k)$; else no moving edge is determined and $CONF(\ell)$ is set to 0.

For each previously selected point ℓ

- If $CONF(\ell) > CONF(\ell_1)$ and $CONF(\ell) > CONF(\ell_2)$, where points ℓ_1 and ℓ_2 are the two neighbours of ℓ in the direction perpendicular to ϕ_k , then

conclude that a moving edge is located at $\ell = \ell$, whose parameters are given by $\hat{\phi} = \phi_k$.

This last step could be interpreted as a thinning procedure. Indeed it corresponds to the local maximization of CRV subject to location parameter ℓ as expressed in (1).

If the volume Π intersects I images, CRV can be decomposed into:

$$CRV(\ell, \phi_j) = \left| \sum_{i \in I} \sum_{m_i \in M_i} a_j(m_i) f(\ell+m_i) \right|$$

$$= \left| \sum_{i \in I} CRV_i(\ell, \phi_j) \right| \quad (5)$$

where M_i is a set of vectorial indices such that $\{\ell+m_i, m_i \in M_i\}$ represent all points belonging to volume Π and image i . Hence, this approach can embrace with the same formalism cases where two and more images are considered.

An additional heuristic is introduced to avoid false detections. Before concluding that a spatio-temporal edge is present at point ℓ according to the criterion (1), the following constraint must be satisfied:

$$\mu_1 \leq |CRV_i| / |CRV| \leq \mu_2, \text{ for all } i \in I \setminus \{1\}$$

where μ_1 and μ_2 are two predetermined thresholds.

More complex modeling, including for instance circle arc or rotation component, could also be handled by the same method. This will only lead to other sets of masks to be considered in expression (4).

One advantage of this method is to present no inherent restrictions concerning kinds of edges likely to be successfully handled (in particular, occlusion boundaries) and concerning measurable motion magnitude. The same does not hold for differential

methods. The extent of measured motion is directly constrained by the smoothing extent used to compute the spatial gradient of the image intensity. Therefore, the differential approach is more appropriate for small displacements. Moreover, flow fields are often incorrect near occlusion, since assumptions required for differentiation do not hold any longer in such areas. On the other hand, this maximum-likelihood technique can be CPU-time consuming with a general-purpose computer, but CPU-time can be drastically reduced if an array processor is used.

III COMPLETE ESTIMATION OF THE VELOCITY FIELD ALONG CONTOURS

In the previous section, a procedure has been described which detects moving edges and, at the same time, estimates their local spatial direction and component of their velocity perpendicular to the contour. The goal of the second stage is to compute component of velocity vectors tangent to the contour.

In order to achieve the second stage of the image flow analysis, edge linking is prerequisite. To this end, only local spatial directions of detected moving edges are taken into account. One-pixel gaps can be filled up. The linking technique is similar to the one presented in [13]. Then, we get a set of contours, i.e., a set of chains of linked spatio-temporal edges.

To compute the entire velocity field along the contours, the second stage of analysis must combine the local measurements yielded by the first stage. This combination stage is efficient if enough variations in spatial orientation occur along obtained contours. For instance, a straight line contour remains a singular case.

Let $\underline{\omega} = (\omega_x, \omega_y) = \left(\frac{dx}{dt}, \frac{dy}{dt} \right)$ be the restriction

of $\vec{V} = \left(\frac{dx}{dt}, \frac{dy}{dt}, 1 \right)$ to the plane (x, y) . Let us

$$\text{consider } e_\ell(\underline{\omega}) = \underline{\omega} \cdot \underline{n}_\ell - v_\ell \quad (6)$$

where $\underline{\omega}$ is the velocity field to be estimated, \underline{n}_ℓ is the unitary vector normal to the local edge element at point ℓ , $\underline{n}_\ell = (-\sin\theta_\ell, \cos\theta_\ell)$, v_ℓ is the measured perpendicular component of velocity at point ℓ . $e_\ell(\underline{\omega})$ is supposed to be a stationary random variable.

Then, the measurement of velocity field $\underline{\omega}$ along a given contour C is formulated as the minimization of the following function:

$$J(\underline{\omega}) = \frac{1}{2} E_C (e_\ell(\underline{\omega}))^2 \quad (7)$$

where E denotes expectation. Motivations for such a criterion can be found in [14]. A stochastic gradient algorithm is used to minimize $J(\underline{\omega})$. The recursive estimation is pursued from one point to its successor in the chain. More precisely, it is expressed as follows:

$$\underline{\omega}_{\ell+1} = \underline{\omega}_\ell - \gamma \cdot \nabla_{\underline{\omega}} e_\ell(\underline{\omega}_\ell) \cdot e_\ell(\underline{\omega}_\ell) \quad (8)$$

where γ is a gain matrix, and ∇ denotes the gradient.

$$\nabla_{\omega} e_{\ell}(\underline{\omega}) = \left(\frac{\partial e_{\ell}}{\partial \omega_x}, \frac{\partial e_{\ell}}{\partial \omega_y} \right), \text{ and}$$

$$\frac{\partial e_{\ell}}{\partial \omega_x}(\underline{\omega}) = n_{\ell}^x = -\sin \theta_{\ell};$$

$$\frac{\partial e_{\ell}}{\partial \omega_y}(\underline{\omega}) = n_{\ell}^y = \cos \theta_{\ell}.$$

The initial estimate can be given by $\omega_0 = v_0^{\perp} n_0$.

A theoretical proof of such a minimization formulation is shown in [15]. The obtained convergence is quite fast. Two recursion cycles around a given contour C are usually sufficient for a proper estimation of the velocity field $\{\omega_{\ell}, \ell \in C\}$. Moreover, two recursions are performed in parallel, clockwise and counter-clockwise. Then, an average is computed between the two estimates at each point ℓ . Smoothness constraint is not explicitly formulated in the minimization criterion as in [16], but it is ensured by the recursion along the contour.

IV RESULTS

IV.1 - Results concerning the first analysis stage

Experiments on computer-generated images have been performed in order to warrant the estimation method of moving edges. Different kinds of motion have been considered (translation of the camera along its view axis, object rotation in the image plane). Results are presented in [12].

The algorithm has also been applied to actual images. Only two successive images are considered for each example reported here. Hence each mask corresponding to coefficients a_j 's for each predefined configuration $\phi_j, j=1, \dots, G$, divides into two submasks. These G masks are computed once G geometric configurations are chosen. Then they are available when images are processed. Choosing angle ψ_j is equivalent to choosing displacement magnitude V_j^{\perp} in the direction perpendicular to the local linear edge element whose spatial direction is θ_j . Therefore, the location of the second submask in the second image with respect to location of the first one centered at current point ℓ in the first image is given by $V_j^{\perp} n_j$.

Then, the G configurations can also be denoted as $\{(\theta_r, \psi_q), q=1, Q, r=1, R\}$ with $R \times Q = G$. Thus the function CRV can be written as follows:

$$\begin{aligned} CRV(\ell, \phi_{rq}) &= \left| \sum_{i \in I} CRVi(\ell, \phi_{rq}) \right| \\ &= \left| CRV(\ell, \theta_r) + \sum_{i \in I \setminus \{1\}} CRVi(\ell, \phi_{rq}) \right| \quad (9) \end{aligned}$$

In order to save CPU-time, convolution operations with the whole set of masks, as previously explained, are not actually computed for each point ℓ . If $CRV(\ell, \theta_r) < a\lambda$, with e.g. $a=0.25$, computations corresponding to the evaluation of $CRV(\ell, \phi_{rq}), q=1, Q$, stop and $CRV(\ell, \phi_{rq}), q=1, Q$ are set to 0.

The first example is extracted from a natural sequence acquired by a camera and depicting an urban scene. Figure 2 shows the first image. The set of masks consists of 66 masks including six possible

spatial directions: $\theta = 0^\circ, 30^\circ, 60^\circ, 90^\circ, 120^\circ, 150^\circ$ and eleven possible perpendicular displacement magnitudes, $V^{\perp} = -5, \dots, -1, 0, 1, \dots, 5$ ($G=66, R=6, Q=11$). Submask size is 5×5 pixels. The estimated perpendicular displacement field is presented in figure 3 along with spatial edges, for an image part.

An evaluation of the correctness of the resulting estimation is available. By means of some other tool, motion is known to be three pixels to the left in the whole image except for subparts corresponding to hung linen, some bushes. Quite satisfactory results are obtained.

The second example includes two images of a printer acquired by a CCD camera (Figure 4). Only the printer has been moved from one image to the next, camera and background remain fixed. 84 masks have been considered, that is to say four possible spatial directions $\theta = 0^\circ, 45^\circ, 90^\circ, 135^\circ$, and 21 possible perpendicular displacements $V^{\perp} = -10, \dots, -1, 0, 1, \dots, 10$. (Of course, metric is adapted when directions other than horizontal and vertical are considered). Determined moving edges are shown in Figure 5 with their perpendicular displacement, which can eventually be none.

IV.2 - Results concerning the second analysis stage

Two sets of experiments are presented involving computer-generated images including a single polygonal object and two kinds of motion: uniform translation and in-plane rotation. Superimposed silhouettes of the object in two successive positions are shown for both cases, respectively in Fig. 6a and 7a. Of course, the method is not restricted to such cases.

In Fig. 6b and 7b is drawn the perpendicular displacement field. It has been estimated using the algorithm presented in this paper. Fig. 6c and 7c show the resulting complete displacement field after two recursive estimation cycles around the boundary. The last example points out that varying displacement field can be successfully handled.

V FUTURE WORK

Future research directions mainly include:

- corner displacement estimation (as complementary processing after linking)
- detection of possible motion boundaries along contours in parallel with the recursive estimation (e.g., this may happen if a boundary portion is an occlusion one) in order to reinitialize the recursion.

REFERENCES

- [1] T.S.Huang (Editor), "Image Sequence Processing and Dynamic Scene Analysis", Proc. of NATO Advanced Study Institute at Braunlage, W-Germ., Springer-Verlag, 1983.
- [2] R.M.Endlich, D.E.Wolf, D.J.Hall, and A.E.Brain, "Use of a Pattern Recognition Technique for Determining Cloud Motions from Sequences of satellite photographs", J^{al} of Applied Meteorology, Vol.10, Feb. 1971, pp 105-117.

The author would like to thank Albert Benveniste for useful comments.

ACKNOWLEDGEMENT

- [3] J.R.Jain, and A.K.Jain, "Displacement Measurement and Its Application in Interframe Image Coding", IEEE Trans. on Communications, Vol. COM-29, n°12, Dec. 1981, pp 1799-1808.
- [4] R.Pagun, and E.Dubois, "A Spatio-Temporal Gradient Method for Estimating the Displacement Field in Time-Varying Imagery", Computer Vision, Graphics, and Image Proc., 21, 1983, pp 205-221.
- [5] J.K.Tsotsos, J.Myllopoulos, H.D.Covey, and S.W.Zucker, "A Framework for Visual Motion Understanding", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-2, n°6, Nov. 1980, pp 563-573.
- [6] P.Rives, and L.Marcé, "Use of Moving Vision Sensors to Control Robots in an Unknown Universe", Proc. of ROVISSEC 5, Amsterdam, Oct. 1985, pp 165-176.
- [7] A.Mitiche, "On kinematics and computation of Structure and Motion", 2d IEEE Comp. Vision Workshop on Rep. and Control, Annapolis, May 1984.
- [8] G.Aliv, "Determining Three-Dimensional Motion and Structure from Optical Flow Generated by Several Moving Objects", IEEE Trans. on PAMI, Vol.7, n°4, July 1985, pp 384-401.
- [9] S.T.Barnard, and W.B.Thompson, "Disparity Analysis of Images", IEEE Trans. on PAMI, Vol.2, n°4, 1980, pp 333-340.
- [10] B.G.schunck, and B.K.P.Horn, "Constraints on Optical Flow Computation", Proc. of Pattern Recognition and Image Proc. Conf., Dallas, Aug. 1981, pp 205-210;
- [11] H.H.Nagel, "Constraints for the Estimation of Displacement Vector Fields from Image Sequences", Proc. of IJCAI, 1983, pp 945-951.
- [12] P.Bouthemy, "Estimation of Edge Motion Based on Local Modeling", Proc. of SPIE Conf. on Computer Vision for Robots, Cannes, Dec. 1985.
- [13] R.Nevatia, and K.R.Babu, "Linear Feature Extraction and Description", Computer Graphics and Image Proc., 13, 1980, pp 257-269.
- [14] P.Bouthemy, "Un Nouveau Schéma d'Estimation du Champ des Vitesses sur les Contours dans une Sequence d'Images", Proc. of CESTA Second Image Symposium, Nice, Avril 1986.
- [15] A.Benveniste, M.Goursat, and G.Rugot, "Analysis of Stochastic Approximation Schemes with Discontinuous and Dependent Forcing Terms with Application to Data Communication Algorithms", IEEE Trans. on Automatic Control, Vol.AC-25, n°6, Dec. 1980, pp 1042-1058.
- [16] E.C.Hildreth, "Computations Underlying the Measurement of Visual Motion", Artif. Intell., 23, 1984, pp 309-354.

Fig.5: Determined spatio-temporal edges with their perpendicular displacement $\lambda=3000$, $\mu_1=0.75$, $\mu_2=1.25$

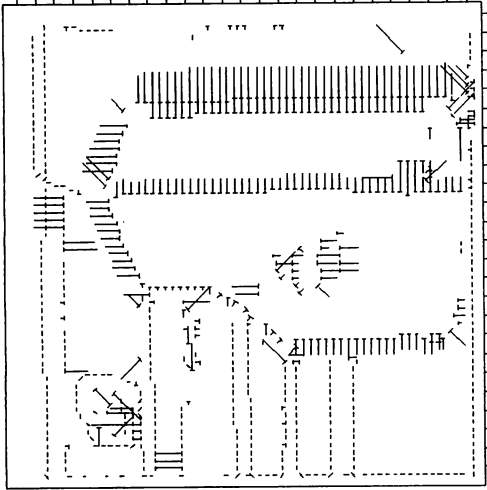


Fig.4: "Printer" sequence

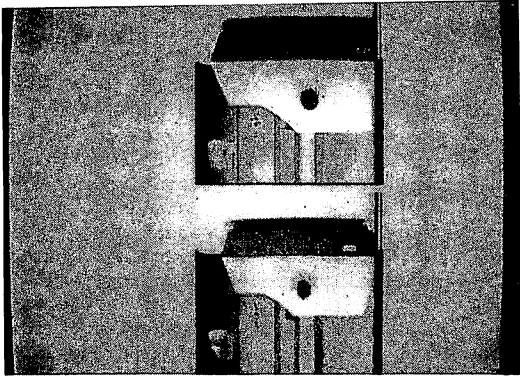


Fig.3: Determined spatio-temporal edges with their perpendicular displacement (axes are sampled each 5 pixels) $\lambda=2500$, $\mu_1=0.8$, $\mu_2=1.2$

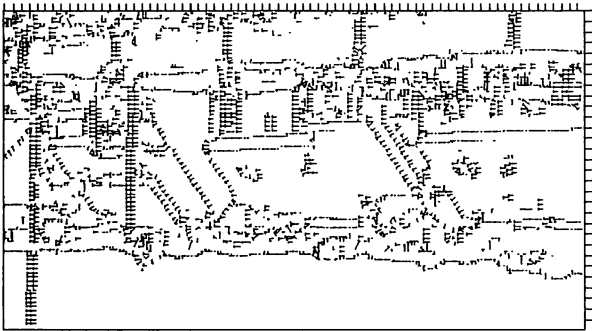
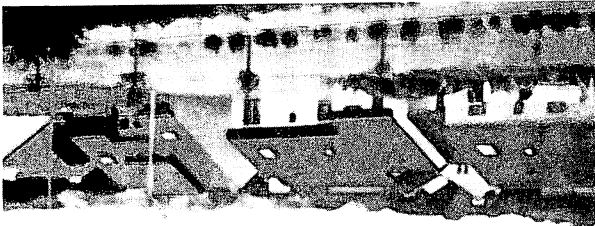


Fig.2: First image of the "house" sequence



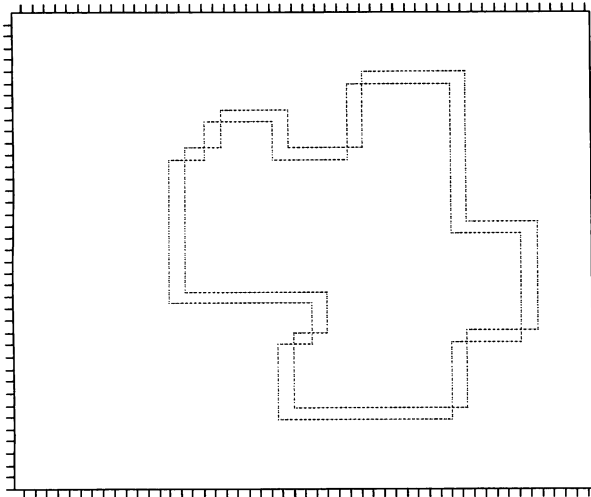


Fig.6a: *Superimposed silhouettes of polygonal object 1*

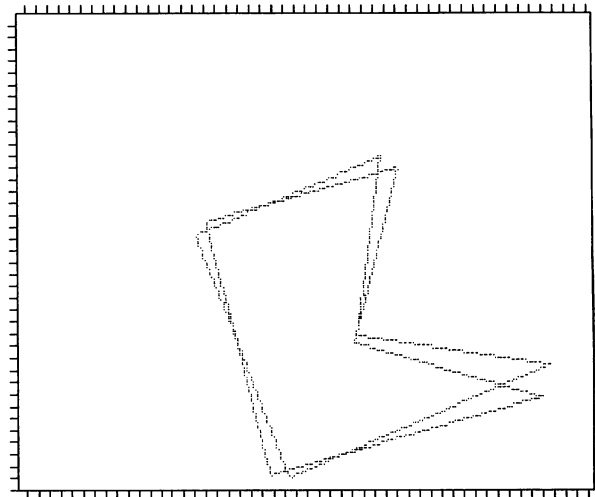


Fig.7a: *Superimposed silhouettes of polygonal object 2*

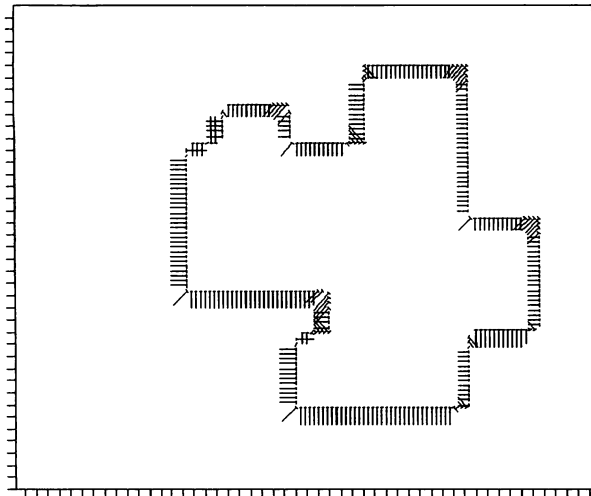


Fig.6b: *Perpendicular displacement field*

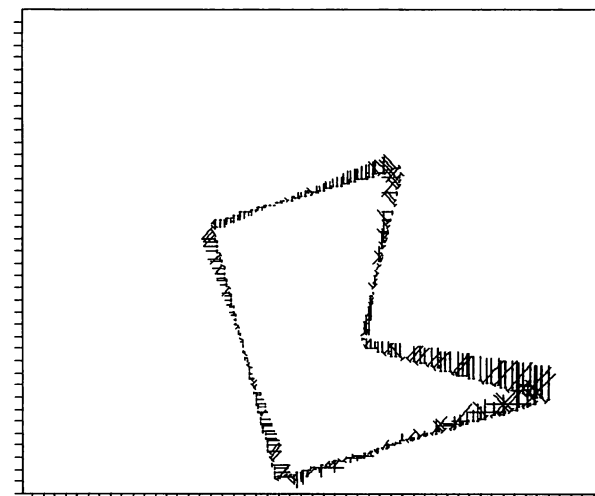


Fig.7b: *Perpendicular displacement field*

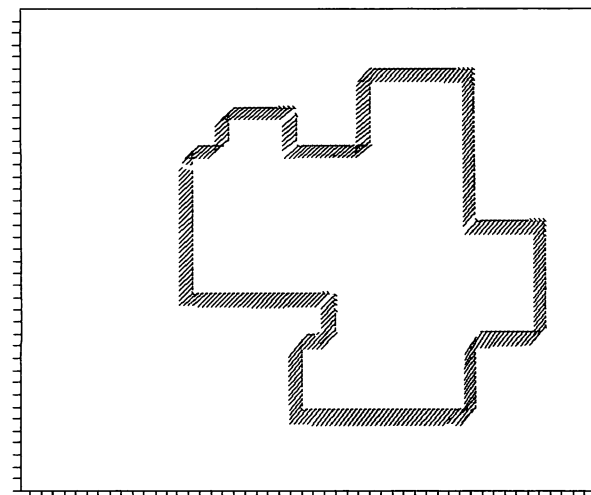


Fig.6c: *Resulting complete displacement field*
 $\gamma = \begin{pmatrix} 0.03 & 0.01 \\ 0.01 & 0.03 \end{pmatrix}$

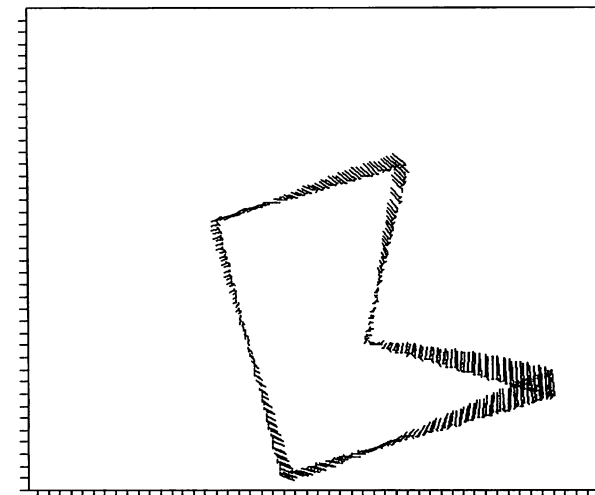


Fig.7c: *Resulting complete displacement field*
 $\gamma = \begin{pmatrix} 0.04 & 0.01 \\ 0.01 & 0.04 \end{pmatrix}$

EDGE-ONLY MATCHING TECHNIQUES IN ROBOT VISION

Shyamala Nagendran and Terry M. Caelli

Department of Psychology, The University of Alberta
Edmonton, Alberta T6G 2E9

Abstract

A computation procedure for detecting arbitrary two-dimensional signals embedded in scenes and independent of orientation/size is developed using pipe-line pixel processor procedures. Signals are encoded as edge-only features and cross-correlated with edge-only versions of the input scenes--both in cartesian and log-polar coordinates. These processes are incorporated into a robot visual system capable of locating, moving towards, and pointing to a target signal, again, independent of its size and orientation.

KEY WORDS: robot vision, pattern recognition, edge extraction, invariance coding

1. Introduction.

Many claims have been made as to the importance of edge information in coding images (Marr, 1982), pattern recognition (Rosenfeld & Kak, 1982) and fast computational vision in general. Secondly, convergent results from human psychophysics (Watt & Morgan, 1983), vertebrate physiology (Pollen, Andrews & Felden, 1978) and computational edge processing (Marr & Hildreth, 1980; Leclerc & Zucker, 1983) point to the "optimal" edge extractor as the logical intersection of band-pass Gaussian filters or pseudo-gamma function band-pass filters (Marr & Hildreth, 1980) approximated by $\nabla^2 G_\alpha$ operators (Watt & Morgan, 1983; Leclerc & Zucker, 1983): a Laplacian operator following low-pass Gaussian filters of specified bandwidths. In this paper we are concerned with using edge information for pattern recognition or pattern matching, as restricted to the two-dimensional environment--though including the problem of matching independent of signal orientation and size--required in our robot pattern recognition system (Figure 1).

Edge-based matching techniques are particularly useful in the context of pipe-line pixel processors where the execution time for convolution operations is critically dependent on kernel size, and the frame memories are restricted to 8- or 16-bit pixel sizes. Though our general model involves the correlation of edge-only pattern features at various levels of resolution according to a strict Laplacian pyramid format, in the present simulations we have restricted our analyses to the highest level of resolution: the original 512x512 input format.

A typical pipe-line pixel processor (Arithmetic Logic Unit: ALU-512) maps frame memories into frame memories with respect to the logical operations of (OR, exclusive OR, AND, 2's complement) and usual addition and subtraction operations. Each pass takes 33 msec and operates on the full image, except when pixel protects are active. Information passes through a 16-bit register and the device also has an 8x8 bit multiplier--which enables convolution



Figure 1

Robot and restricted visual environment used in simulations. Various image montages were placed about the walls and the robot's task was to detect where a specified signal was, move towards and point to it, based on visual information.

operations on 512x512 images with $n \times m$ -sized kernels, and so taking $n \times m \times 33$ msec per convolution.

The robot was controlled by interrupt-driven subroutine calls capable of moving it in any horizontal direction and operating a three-joint arm according to the processed image information, as shown in Figure 1. Both image processing (Imaging Technology) and robot (RB Robots, Colorado) systems were controlled by a PDP 11/23 computer operating in RT11.

2. Edge-extraction, invariance codes, and matching techniques.

As implied above, the $\nabla^2 G(\alpha)$ operator (on an image I) is an adequate representative of band-pass filtering used in recent edge-extraction techniques. Defined by

$$\nabla^2 G(\alpha)(I) = \frac{\nabla^2 G(\alpha; x, y)(I)}{\partial x^2} + \frac{\nabla^2 G(\alpha; x, y)(I)}{\partial y^2} \quad (1)$$

$$\text{where } G(\alpha; x, y) = e^{-\alpha^2((x-x_0)^2 + (y-y_0)^2)} \quad (2)$$

Here (x_0, y_0) defines all convolution centers over the image. This filter is simple to approximate in a pipe-line pixel process by two convolution kernels. First, the Gaussian low-pass filter (2) can be approximated by the recursive use of an averaging kernels of the form:

$$A = 1/4 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad (3)$$

After n recursions, it produces the bivariate binomial coefficients on an $(n+1) \times (n+1)$ kernel as:

$$G \approx B(n; x, y) = \frac{(n!)^2}{x! y! (n-x)! (n-y)!}, \quad x, y = 0, \dots, n \quad (4)$$

This takes $n \times 132$ msec to complete and in the simulations to be reported here n was set at 10 (a total of $33 \times 4 \times 10 = 1.32$ sec).

The ∇^2 or Laplacian kernel is defined (in finite difference form²) by

$$\nabla^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (5)$$

taking $9 \times 33 = 297$ msec to complete.

Though various options are available for extracting the image edges as (x, y) coordinates from the $\nabla^2 G$ images, we have used the following technique, since it only involves two passes through the pipe-line pixel processor. Since the output of the $\nabla^2 G(\alpha)$ operator can be positive or negative, 127 was added to the output, followed by an .AND. with 128, giving the complete form:

$$\{E(x, y)\} = \{all\ z(x, y) > 0, \text{ from } [G_2[\nabla^2 G_{10}(I) + 127]] \wedge 128\} \quad (6)$$

Here, also, a second smoothing operation (G_2) was employed to suppress isolated non-zero points resultant from the $\nabla^2 G_{10}$ operation. An illustration of these processes is shown in Figure 2, taking a total time of 1.75 sec. In our application the edge set of points (6) was (spatially) uniformly sampled to less than 256 points (rectangular area shown in Figure 2a), as shown in Figure 4b.

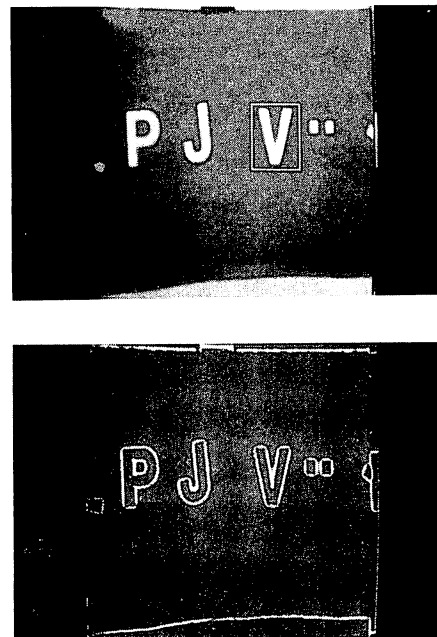


Figure 2

(a) Input image (I) and (b) edge-only version created by: $G_2(\nabla^2 G_{10}(I) + 127) \wedge 128$ (rectangular area in (a) corresponds to signal used in matching process, Figure 3).

Once this edge code has been established for a given signal, cross-correlation of the signal with any arbitrary scene can be reduced to a number of passes through the pipe-line processor equal to the number of signal points. That is, the cross-correlation function

$$C_{sg}(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} s(\alpha,\beta)g(x+\alpha,y+\beta)d\alpha d\beta \quad (7)$$

reduces to $C_{sg}(x,y) = \sum_{n=x_h}^{x_l} \sum_{m=y_l}^{y_h} S(n,m)g(x+n,y+m), \quad (8)$

(x_l, y_l, x_h, y_h) corresponding to the sub-array containing the signal. For both s and g being binary valued (edge-only) images (8) reduces to

$$\hat{C}_{sg}(x,y) = \sum_{n,m \in E(x,y)} s(n,m)g(x+n,y+m). \quad (9)$$

This correlation function can be executed by adding the image g to itself shifted by the locus of points (x,y) values depicting the signal, or

$$\hat{C}_{sg}(x,y) = \sum_{n,m \in E(n,m)} g(x+n,y+m) \quad (10)$$

since $s.g=1$ if, and only if, both signal and image edge components are present. (10) is readily executed via a series of passes through the pipe-line pixel processor where the results are accumulated--as illustrated in Figure 3b.

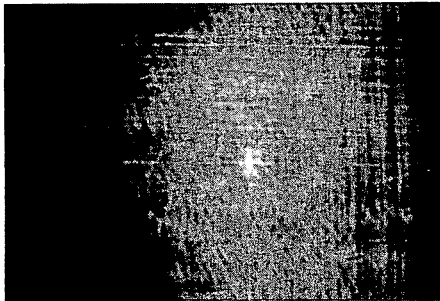


Figure 3

Edge-only matching (cross-correlation) between sampled signal (a: see Figure 2b) and image (b). Luminance corresponds to the likelihood of signal presence.

The problem with this matching technique, however, is that it is not invariant to size and orientation changes of the signal in the image. This is a particularly relevant problem in the case of robot vision studied here since movements of the robot towards the field wall introduces size and possibly small angular changes in the signal (Figure 1).

Techniques for matching which overcome rotations of signals have been recently developed by Hsu et al.⁷ using circular harmonic functions whereby a Fourier decomposition of the polar transformed signal and rotated version is enacted along the orientation parameter. Matching occurs since both images have identical power spectra, with the phase giving the orientation differences between them. The problems associated with these techniques are (1) determining the initial estimate of the signal and image centers to enact the polar transforms, and (b) that the Fourier transform needs to be computed.

An alternate approach to the problem is simply to transform edge-only images into log-polar coordinates and enact matching using pixel processor operations. Again, this process has the problem of estimating the image center for the log-polar transform. For both signal and image the best *a priori* initial estimate of the center is the peak of the cartesian cross-correlation functions; though we are at present investigating a pyramid search technique to improve the accuracy and the speed of estimating the log-polar centre.

The log-polar (conformal) mapping is:

$$r' = 77.8 \log_e(r_0 + 1) \quad (11)$$

$$\text{and } \theta = \tan^{-1}(Y - Y_0 / X - X_0)$$

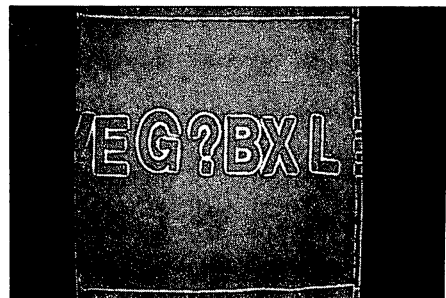
where

$$r_0 = \sqrt{(x - x_0)^2 + (y - y_0)^2} \quad (12)$$

for (x_0, y_0) corresponding to the peaks of signal autocorrelation and image cross-correlation images.

Figure 4 shows an example of signal and images transformed by the above procedure. Here, the peak of the log-polar cross-correlation function determines the particular signal size and orientation detected--with respect to the center chosen by the cartesian cross-correlation procedure.

(a)



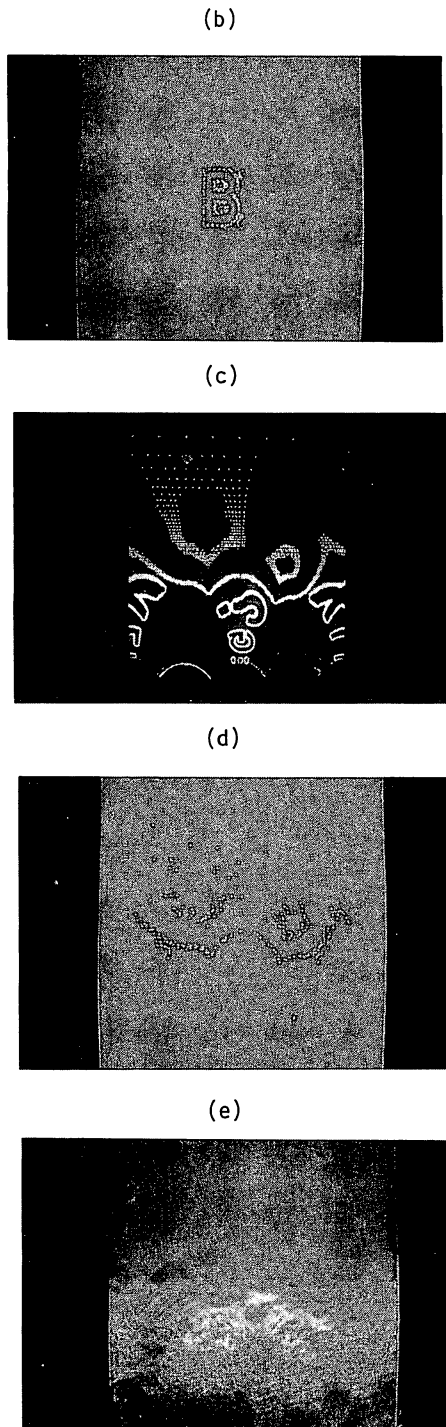


Figure 4

(a), (b) show cartesian edge-only image and sampled signal respectively, while (c), (d) show corresponding log-polar versions of (a), (b). (e) Shows log-polar cross-correlation image where the peak defines the orientation and size of the signal embedded in the size which best matches the input signal.

Though the basic computational vision procedures are defined and illustrated, a number of related problems to implementing these ideas into a pseudo-real time robot visual system must be resolved--particularly related to the analysis of error, setting of thresholds and adaptation procedures. Even in the restricted environment used in these simulations ambient luminance introduces photon fluxuations such that even redigitizing exactly the same image does not necessarily result in identical pixel values, albeit they are close. Of particular variability is the luminance projections around the field walls about which the robot moves.

For these reasons a signal matching threshold was set by digitizing the signal and then redigitizing the same area and determining the edge-only cross-correlation function. The peak value could then be used to estimate the signal match threshold--which in these simulations was set at 60% the number of signal points. Due to the rounding errors associated with the log-polar transform, the polar signal matching threshold was typically set at two-thirds that of the cartesian. Although these thresholds seem remarkably low, the checking procedure resulted in no false alarms in our restricted simulations--though more detailed signal detection analysis is being pursued.

5. Conclusions.

In these simulations we have demonstrated how pipe-line pixel processor technology may be effectively employed to enact edge-only matching of arbitrary signals to images. Though to this stage the choices of thresholds are arbitrary, our results using the log-polar mapping procedure for matching under rotations and size change along with the cartesian procedure resulted in successful matches for alphabetic characters and relatively complex images.

This log-polar matching procedure falls down when no evidence of a match occurs--that is, no discernible peak occurs in the cross-correlation function. In other attempts to solve this problem, an adaptive procedure is used to establish the center. However, this is time consuming in that the procedure involves changing between polar and cartesian coordinates. The second limitation of the above procedure is the simple use of signal sampling to keep the number of signal pan and scroll values (coordinates) below 255. This does not necessarily result in the signal features which are more likely to

remain invariant to light fluxuations, image distortions, etc. However, these problems are currently under investigation, and the matching procedure and criteria are being analyzed from a signal detection perspective. Indeed, if pipe-line pixel processes could enact log-polar cross-correlations for every possible center, then these problems would be solved.

References

1. Marr, D. (1982). Vision. San Francisco: Freeman.
2. Rosenfeld, A. & Kak, A. (1982). Digital Picture Processing: Vol. 1. New York: Academic Press.
3. Watt, R. & Morgan, M. (1983). The recognition and representation of edge blur: evidence of spatial primitives in human vision. Vision Research, 23, 1465-1477.
4. Pollen, D., Andrews, B. & Felden, S. (1978). Spatial frequency selectivity of periodic complex cells in the visual cortex of the cat. Vision Research, 18, 665-682.
5. Marr, D. & Hildreth, E. (1980). Theory of edge detection. Proceedings of the Royal Society (London), B207, 187-217.
6. Leclerc, Y. & Zucker, S. (1983). The local structure of image discontinuities in one dimension. Technical Report TR-83-19R, Department of Electrical Engineering, McGill University.
7. Hsu, Y., Arsenault, H. & April, G. (1982). Rotation-invariant digital pattern recognition using circular harmonic expansion. Applied Optics, 21(22), 4012-4015.

A METHOD OF LEARNING RULES FROM UNCERTAIN DATA
APPLIED TO THE COMPUTER VISION PROBLEM

D. Hutber and P. Sims
Sowerby Research Centre (FPC 267)
British Aerospace PLC, Naval Weapons Division
P.O.Box 5, Filton, Bristol BS12 7QW, England

ABSTRACT

The construction of knowledge-based systems of a size large enough to be useful has led to problems of knowledge acquisition. A way of solving this is to enable the computer to automatically generate its own knowledge from sets of sample data. This becomes further complicated when the sample data may have errors or noise in it.

This paper describes a system that generates knowledge in the form of rules from uncertain data, in the domain of computer vision. The way in which the uncertainty arises and is processed is discussed, and some sample results are presented.

KEYWORDS: machine learning, fuzzy sets, computer vision.

INTRODUCTION

The construction of knowledge-based systems of a size large enough to be useful has led to problems of data acquisition. Expert systems have relied on the interaction between a knowledge engineer and a domain expert to produce a set of rules that capture the expert's knowledge on a particular topic. This process is very time-consuming, and as the size of the knowledge base increases it becomes a limiting factor. In computer vision, this method has the additional problem that the language necessary to represent the rules is not well-defined. The information given to any vision system is usually in the form of pixels, but formulating rules in terms of pixels is computationally expensive and would be difficult for a programmer to understand. A higher-level representation language is required in order to bring down the computation cost and to aid comprehension.

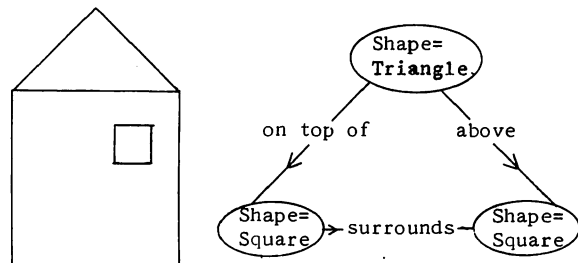
This paper addresses the subject of 'machine learning from examples', or equivalently of automatically generating rules to describe a concept from examples and counter-examples of that concept. Desirable properties of such a generation process are ease of inclusion of additional problem-specific knowledge, and ease of comprehension by a user or programmer. The representation of the examples and rules is hence of primary importance, since to a large extent this will determine the

range of situations that can be expressed, and the manipulations it is possible to perform.

The problem of interpreting uncertain data has received considerable attention from people building expert systems, e.g. MYCIN (Shortliffe Buchanan 1), but the problem of learning rules to describe uncertain data has been studied less. In computer vision there is uncertainty due to imperfect image processing and noise. Here this has been modelled by the technique of fuzzy sets.

EXAMPLES AND COUNTER-EXAMPLES

Objects are made up of sub-objects called 'primitives'. The primitives have properties that are called 'attributes', and there are connections between the primitives which are expressed as relations. For the purposes of computer vision, these primitives are the regions, and the attributes may be properties such as shape, size and colour; the relations are 2-D spatial relationships such as 'above' or 'surrounds'. This representation corresponds to a semantic net or graph.



Here shape, size and height are the unary descriptors used and these take values of, for example, shape=triangle, size=medium and height=6. This illustrates the use of two types of unary descriptors:

- . nominal descriptors, where the values have names.
- . linear descriptors, where the values are numbers.

The two types of unary descriptor are treated in different ways. More restrictions are placed on linear descriptors since it is assumed that

they are ordered in a meaningful way, and that integer values differing by a small amount give rise to similar properties and can be grouped together.

Nominal descriptors have discrete names with no ordering implied on them. For example, it is not meaningful to describe a shape as halfway between a circle and a square, although as will be shown later, it is possible to express equal uncertainty as to whether a primitive's shape is 'circle' or 'square'.

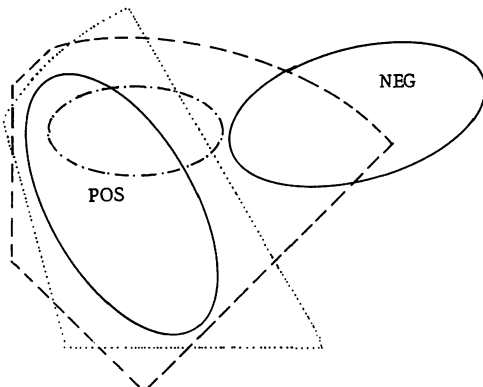
The only binary descriptor used at present is 'is spatially related to'. This takes values of, for example, 'surrounds' and is a nominal descriptor. Each value has an inverse, e.g., 'is-surrounded-by'.

GENERAL METHOD

The central idea in the learning algorithm described here is one of 'generate and test'. The method is an extension of the INDUCE algorithm (Michalski 2) where a series of trial descriptions is generated using a 'seed' example, and tested against examples and counter-examples. The seed example provides the descriptors from which the trial descriptions are constructed. After a description has been tested, if it is ranked better than those before it in the series, according to some criterion, it is retained and used to produce several more descriptions. The new descriptions are produced by specialising the old description. If it is no better than those before it, the trial description is discarded.

This guided generation process is equivalent to a search. The search is over a space of all possible descriptions, consisting of properties of, and relations between sub-objects, and this is guided by a set of examples of a concept and a set of counter-examples. These representatives are very important to the working of the algorithm, and so a good choice of examples, and more critically counter-examples, is essential (Winston 3).

In simple set terms, if we consider a space of all possible objects, then we may represent POS, the set of examples, and NEG, the set of counter-examples, by the sets shown in the figure below:



A trial description will cover a number of possible objects. Three descriptions are

represented by the sets in dashed lines, each of which has a different property. The set ---- covers all the examples and is called a 'complete description', and the set ---- covers none of the counter-examples and is called a 'consistent description'. The aim of the learning algorithm is to produce a number of complete and consistent descriptions, for example, the set ----, which can then be used on unknown objects to identify them as members of the concept.

The difference between this type of learning and a decision tree is the inductive process, whereby descriptions of a class of objects are induced from the sets of examples and counter-examples. The induction in this case is performed by generalisation rules which act on a description to produce a more general description. In terms of the set diagram above, generalisation increases the range of objects that the description covers.

GENERALISATION RULES

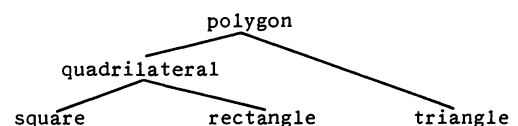
Generalisation is performed only on consistent descriptions generated by the learning algorithm (i.e. those descriptions which do not cover any of the counter-examples). The reason for this is that the aim of the algorithm is to produce a series of consistent descriptions that are as simple as possible. Hence when a consistent (but not complete) description is produced, it is generalised, hoping that the new description will cover more examples in POS whilst maintaining the consistency property.

There are really only two generalisation rules used in the implementation, and they correspond to internal disjunction of values of the two types of descriptor. They are:

- (i) Adding alternative (or range of alternatives).
- (ii) Closing interval.

(Only the first of these will be described in detail.)

(i) The adding alternative rule works on nominal descriptors, using two values of the same descriptor, one of which is in the description already, and the other which it is desired to include. Values of the descriptor may already be grouped together, either by the programmer or by the system when it has learnt a rule in the past, to form a structure. The existing structure of the descriptor is searched to find all existing groupings of values that include both of the required values. For example, the 'shape' descriptor may have the structure shown below:



Here 'quadrilateral' is defined as 'square or rectangle' and 'polygon' as 'quadrilateral or triangle' by the user of the program. The structure is in this case a tree which can be used as a convenient means of generalisation, so that square

is-a-kind-of quadrilateral is-a-kind-of polygon. However, this restricted generalisation has the disadvantage that the user has to supply all the structure, and that if the combination 'rectangle or triangle' (but not square) appeared repeatedly this could not be expressed efficiently. An alternative structuring technique adopted by this work is a 'group if useful' technique, where initially the user can specify as much or as little grouping as he sees fit, and the program will group together values if it repeatedly finds such a process useful. Thus if no structure were supplied to the 'shape' descriptor, but the combinations 'square or rectangle' and 'rectangle or triangle' occurred frequently as the algorithm ran, then the structure of the descriptor would look like the figure below:



The language being used here is clearly less comprehensible than that used in the previous structure, but for display purposes the original 'square or rectangle', etc. may be used. It has the advantages of being easier to manipulate and being able to express a wider variety of combinations of values.

Each of the groupings containing the two values is used to form a generalised description which is tested for consistency, starting with the largest grouping (corresponding to the most general description) and going on to the smallest. When a consistent generalised description is found the process stops. It is only required to test this series of descriptions on the counter-examples to establish the consistency property. If no consistent generalised description is found, a group consisting of the two values only is created and the corresponding description is tested. If this fails, the two-value group is deleted and generalisation of this descriptor is abandoned.

(ii) The closing interval rule works in much the same way on linear descriptors, using intervals including the two values rather than on groupings.

STRUCTURE AND ATTRIBUTES

The way that the algorithm is implemented is to process the binary descriptors first, generating structure-only descriptions. Each of these structure descriptions is then used as a framework in which to run the algorithm for the unary descriptors. There are two effects arising from the separation of the unary and binary descriptors. These are:

- (1) Since structure is treated first, the algorithm preferentially generates solutions with structural conditions rather than attribute conditions.
- (2) Any description obtained with a consistent structural part will be consistent.

PREFERENCE CRITERIA

The progression of the learning algorithm is influenced by two separate preference criteria. These are now described, and their effect on the type of description generated outlined.

Preference Criterion (1):

This is used on every description in order to quantify how close to being a solution it is. The measure used is simply:

Number of examples in POS covered by description -
number of counter-examples in NEG
covered by description.

Preference Criterion (2):

The preference criterion used here is a cost function which states how successfully a description satisfies certain requirements. It is evaluated as a weighted sum of the length, cost, and degree of generalisation of a description. These weights are user defined according to the type of description it is wished to generate (e.g. long and specific or short and general). The contribution from each characteristic will be represented by a number between 0 and 1, defined as follows:

- (i) Length of description
- (ii) Cost of generating description
- (iii) Degree of generalisation of description

The features used in this preference criterion are not exhaustive; for example, in a more complex system, computational simplicity, least possible memory used in storage and overall comprehensibility may be important characteristics for a description to exhibit.

UNCERTAINTY

The main difference between this system and those previously implemented is the way the quality of data relating to examples is treated. For example, a square might be a perfect example of a certain concept, but due to the imaging system it may not actually have a representation that exactly satisfies the axiomatic requirements for a square. Nevertheless it may have a certain perceptual similarity to a square, and may well be one in the actual scene which has become distorted in the imaging system.

There are several alternatives for representing uncertainty. In the majority of systems, Bayesian Probabilities have been favoured; however, Fuzzy Sets and the Shafer-Dempster approach (4) have also received attention in recent literature. Fuzzy sets (Zadeh 5) were selected to represent the uncertainty in the system.

Each fact is assigned a Fuzzy Truth Value (FTV) from 0 to 1. This value represents the degree of membership of the fact in the fuzzy set of true facts. Hence, a description which matches a series of facts from an example or counter-example

will have a list of FTVs associated with it. These are then combined to give an overall fuzzy truth value for the description. If the description is made up of n descriptors, and the j th descriptor matches a fact in a specific Example with FTV ψ_j , then the FTV of the entire description (or the Degree of Fit of a description to an Example, E) is defined as:

$$F(E) = 0.5 + \frac{1}{n} \left\{ \sum_{\psi_j > 0.5} \frac{(\psi_j - 0.5)^2}{0.5} - \sum_{\psi_j \leq 0.5} (0.5 - \psi_j) \right\} \quad (1)$$

The function $F(E)$ has the following properties:

- (i) Simple polynomial form.
- (ii) Sensitive to all truth values (unlike MAX or MIN).
- (iii) Independent of order of truth values.
- (iv) Facts with FTVs < 0.5 are given greater weight in the calculation than those with FTVs > 0.5 .
- (v) $F(E) < \psi_j$ for $n=1$, $0.5 < \psi_j < 1$.

EFFECTS OF THE INTRODUCTION OF UNCERTAINTY

The definitions of Consistency and Completeness now become dependent on the degree of fit. A consistency threshold $T_{\text{consistent}}$ is set such that a description will not be consistent if $F(CE) > T_{\text{consistent}}$ for any counter-example CE . A completeness threshold T_{complete} is also set. If $F(E) > T_{\text{complete}}$ for an example E then that example is defined to be covered by that description.

The introduction of uncertainty into the definitions of Consistency and Completeness affects the evaluation of the Preference Criteria. With Preference Criterion (1) the definition is unchanged except that the number of examples in POS covered by the description will be those examples with degree of fit greater than the completeness threshold. Similarly, the examples covered in NEG covered by a description will be those with degree of fit greater than the consistency threshold.

Preference Criterion (2) is affected by the introduction of two new factors: the consistency and completeness ratings of a description, defined as follows:

- (i) Consistency of description

If the consistency threshold is exceeded by any counter-example then the consistency condition is broken and the consistency rating is set to zero. If the degree of fit for the i th counter-example is $F(CE_i)$ ($i=1..n$) then:

$$\text{Consistency Rating} = 1 - \frac{1}{n} \sum_i \frac{F(CE_i)}{T_{\text{consistent}}} \quad (2)$$

(N.B. If $F(CE_i)=0$ for all i then Consistency Rating=1)

- (ii) Completeness of description

If the degree of fit for the i th example is $F(E_i)$ ($i=1..n$) then:

$$\text{Completeness Rating} = \frac{1}{n} \sum_i \frac{F(E_i)}{1} \quad (3)$$

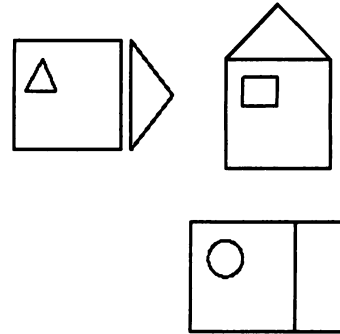
$F(E_i) > T_{\text{complete}}$

(N.B. If $F(E_i)=1$ for all i then Completeness Rating=1)

Hence, the evaluation of the Preference Criterion now becomes a weighted sum of five features: length, cost, degree of generalisation, consistency, completeness. The introduction of Completeness and Consistency ratings has two effects in guiding the system. By weighting in favour of completeness the system can be biased to include all positive examples. By weighting in favour of Consistency the system can be biased against including any counter-examples.

RESULTS

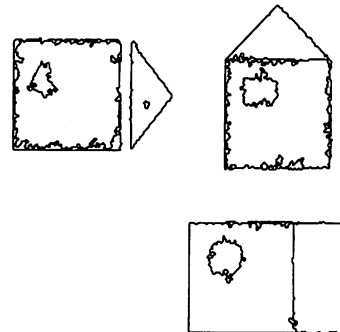
This section shows the results of running the algorithm on a synthetic image, before and after adding Gaussian noise to it. The different descriptions generated in each case are given:



Processed Version of Perfect Input Data.

Rule Generated:

'There are two objects X and Y such that
(X surrounds Y and
Y is a square or a circle)'



Processed Version of Imperfect Input Data.

Rule Generated:

'There are three objects X,Y and Z such that
(X surrounds Y and
Y is a rectangle) or
(X is right of Y and
X is right of Z and
X is a rectangle)'

The result of adding the noise is that the surrounded objects in the examples cannot be reliably labelled as a square and circle as before. The object in the top example is now considered more likely to be a rectangle and the surrounded object in the bottom example is too degraded to be incorporated as part of a rule. This results in the second half of the above rule being generated.

DISCUSSION

In its present form, the learning system described has several problems associated with it, due to the incorporation of uncertainty in the algorithm. Some of these problems are described below.

Coverage of Seed Example.

The INDUCE algorithm is guaranteed to give a solution and terminate eventually (when working on noise-free data), even if the rule obtained is a disjunctive list of the examples in POS. (In this case, no induction has been performed by the system.) The reason why the algorithm terminates is because all the descriptions generated using a 'seed' example are partial descriptions of the example and hence cover it. As the algorithm builds up longer partial descriptions of the seed example, the set of objects covered by the description become smaller, until eventually only the one example is covered.

The use of the degree of fit measure defined above means that partial descriptions of the example will not necessarily 'cover' (in the fuzzy sense) the seed chosen. A consequence of this is that the algorithm cannot be guaranteed to give a solution, unless some other constraints are placed on it. If the situation occurs in which the seed example may not be described without a counter-example also being covered, then to all intents and purposes the descriptors chosen do not discriminate between this example and the counter-example. This may be remedied in one of two ways:

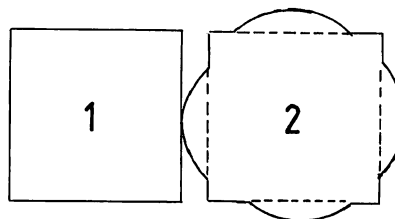
- (i) Alter the Tcomplete threshold to discover whether any setting will give discrimination.
- (ii) Use a better set of descriptors.

Degree of Fit Measure.

The degree of fit measure as defined in Equation 1 has the property that if two facts in an example with truth values of 0 and 1 respectively were matched to two descriptors making up a description, then that example would have a degree of fit of .5 to that description, in spite of the FTV of 0 which is designed to represent the falsity of that fact. This is resolved by making the additional assumption that if the degree of membership of a fact is less than a threshold value T (e.g. $T=0.3$), then it is deleted from the data base. This can prevent the matching of low membership facts and cut down the processing done by the algorithm.

Interdependence of Certainty Values.

This problem is perhaps best illustrated by an example. Consider the two primitives in the figure below:



If primitive 2 is a square, then it is not touching primitive 1.

If primitive 2 is a circle, then it is touching primitive 1.

In other words, the certainty of the relation '1 is touching 2' is dependent on the interpretation of the shape of primitive 1. It is therefore assumed for simplicity that the facts describing the examples are independent of each other, to an approximation.

CONCLUSIONS

In this work, a machine learning scheme for computer vision that models the effects of introducing uncertainty has been implemented. At present, this work is at an early stage and has only been applied to simple, synthetic image data to investigate the changes that occur when uncertainty is present. From the results obtained to date, it seems that the rules that are learnt from perfect data may differ significantly from those obtained from the imperfect equivalent.

ACKNOWLEDGEMENT

This research was supported in part by the Alvey Directorate as part of the Alvey project 'Identification of Objects in 2-D Images'.

REFERENCES

1. Shortliffe E.H. and Buchanan B.G. 'A Model of Inexact Reasoning in Medicine'. Math. Biosci., Vol. 23, 1985.
2. Michalski R.S., Carbonell J.G., Mitchell T.M. eds. 'Machine Learning: An Artificial Intelligence Approach'. Palo Alto: Tioga, 1983.
3. Winston P.H. 'Artificial Intelligence'. Addison-Wesley, 1984.
4. Shafer G. 'A Mathematical Theory of Evidence'. Princeton University Press, 1976.
5. Zadeh L.A. 'Fuzzy Sets'. Information and Control, Vol. 8, pp 338-353, 1965.

SEQUENTIAL ESTIMATION OF BOUNDARIES IN TEXTURE IMAGES

Dragana Brzakovic
Dept. of Electrical Engr.
The University of Tennessee
Knoxville, TN 37996-2100

Antonios Liakopoulos
System Dynamics Inc.
1219 N. W. 10th Ave.
Gainesville, FL 32608

ABSTRACT

In this paper we describe a method for estimating boundaries between perceptually distinct regions in an image. The method is a two step procedure which first identifies image regions which exhibit uniformity. Boundaries between uniform regions are approximated by a complete cubic spline function and the linear recursive filter is used to estimate the values of the function at the knots. Boundaries are assumed to be smooth; however, abrupt changes in boundary location may infrequently occur. In these exceptional cases flexibility of boundary approximation is achieved by adding additional knots at the positions where abrupt changes occur.

KEYWORDS: segmentation, splines, texture, linear recursive filter.

1. INTRODUCTION

Determination of boundaries delimiting objects and their parts in an image is recognized to be a crucial link between an image and its interpretation. It is well recognized that boundary detection in images wherein meaningful regions exhibit textural properties is a difficult task. Regions in such images can be identified by region based segmentation operators, e.g. [10,11]. However, determination of boundaries between texture regions requires further processing. Recently, attempts have been made to develop estimation theory-based boundary detectors [1,4].

This paper describes the design, implementation and performance of an estimation theory-based segmentation operator for noisy images which contain regions of uniform intensity as well as texture regions. The operator performs segmentation at the signal level and it assumes no a priori knowledge on images considered. Its essence is incorporation of region based segmentation with curve fitting to find boundaries between perceptually distinct regions in an image. First, dominant regions which exhibit uniformity and which are called cue regions are identified. Then, boundaries lying between cue regions are estimated globally. Boundaries are approximated by a complete

cubic spline function. The flexibility of boundary approximation is achieved by adding additional knots at positions where boundary location changes abruptly.

The function of this segmentation operator corresponds to early vision in humans and it is viewed as a part of a multi-level modular segmentation scheme. At the first level of segmentation hierarchy this operator is employed to perform rough segmentation of an image. The obtained result is then used as an input to subsequent levels of segmentation which take advantage of knowledge on the scene domain and are task dependent. Their function is to perform refined segmentation, label cue regions and identify smaller objects that are of interest.

The method underlying identification of cue regions is described in Section 2. Boundary approximation and estimation schemes are the subject of Section 3. Results and future work are discussed in Section 4.

2. IDENTIFICATION OF CUE REGIONS

The first task of the segmentation operator is to identify cue regions and intermediate zones where boundaries between cue regions lie. Consequently, the problem of boundary estimation between regions of unknown properties reduces to the problem of boundary estimation in the ambiguity zone between two regions of known properties.

Since most of textures appearing in nature are viewed as "uniform" only as a whole while locally they exhibit various statistical and structural irregularities images are subjected to pre-processing prior to region identification. The purpose of this procedure is to eliminate minor textural detail and map texture regions into regions which exhibit higher degree of ergodicity Figure 1(b). This task is accomplished by using Gaussian filter applied locally over a pixel neighborhood and as a result local image variances decrease as a function of σ . The filter is implemented by using method of hierarchical discrete correlation [3] which performs filtering in stages and approximates Gaussian by weighted sums over small neighborhoods.

The identification of uniform regions involves generation of a T image in which each pixel (x,y) is

replaced by

$$T(x,y) = \frac{1}{(2P+1)(2Q+1)} \sum_{q=-Q}^Q \sum_{p=-P}^P \left(I(x,y) - I(x+p, y+q) \right)^2$$

Generation of a T image is a form of primitive segmentation since $T(x,y)$ is constant within a homogeneous region and increases at the boundary between two homogeneous regions. (Examples of T images are shown in Figure 1 (c)). Cue regions and ambiguity zones where boundaries between cue regions lie can be easily extracted from a T image using thresholding techniques. Results obtained by using method proposed by Otsu[9] are shown in Figure 1 (d).

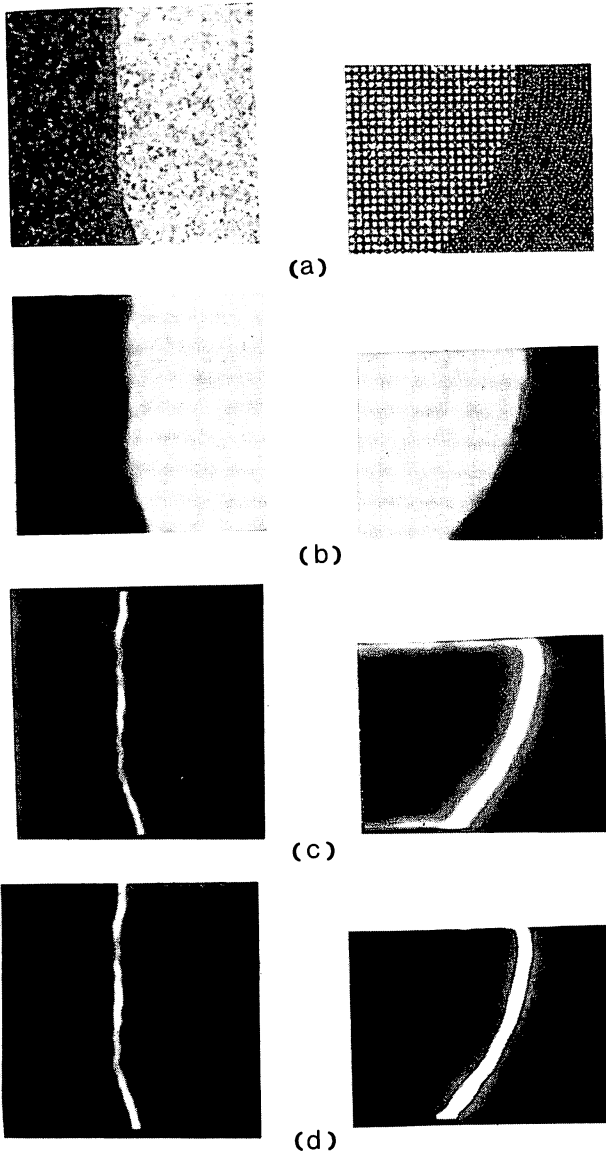


Figure 1. Cue region identification:
(a) original images, (b) filtered images,
(c) T images, (d) identified ambiguity zones.

In essence, described procedure is a region merging method where intensity in the T image is used as the merging criterion. However, it does not employ region splitting; instead, pixels which can not be assigned to cue regions constitute ambiguity zones where boundaries between cue region lie. This design decision is governed by the fact that this operator aims at estimation of boundaries between dominant and thus large regions which exhibit statistical homogeneity. Furthermore, reliability of statistical measurements for texture regions falls with decrease of region size.

3. BOUNDARY ESTIMATION

A boundary $f(y)$ (Figure 2) is approximated, as in an earlier work [1], by a cubic spline function $g(y)$ with $(n+1)$ knots. The function $g(y)$ is explicitly determined by the function values at the knots and derivatives at the interval ends, i.e.

$$g_i(z) = \sum_{m=1}^{n+1} K_{m,i}(z) f'_m + \Lambda_i(z) f'_1 + \Omega_i(z) f'_{n+1} \quad (1)$$

The task of boundary estimator is then to estimate f_i, f'_i $i=1,2,\dots, n+1, j=1, n+1$. This task is accomplished by the linear recursive filter [7] under assumption that state vector evolves according to

$$\underline{x}_k = \underline{x}_{k-1} + \underline{w}_k,$$

\underline{w}_k is the process noise and

$$\underline{x}^T = [f_1, f_2, \dots, f_{n+1}, f'_1, f'_{n+1}].$$

Vector \underline{x} is estimated by considering M simultaneous measurements in the ambiguity zone between two uniform regions. The measurement model takes form

$$\underline{z}_k = H_k \underline{x}_k + \underline{v}_k \quad (2)$$

where H_k is the measurement matrix and \underline{v}_k is the measurement noise. A measurement performed in the ambiguity zone (in pre-processed image between regions R_1 and R_2) is modeled as

$$z = \frac{p_1 \delta + (p - 2\delta) p_2}{\rho} + \eta \quad (3)$$

where p_1 and p_2 are measured properties of regions R_1 and R_2 respectively, η is the measurement noise and ρ and δ are as shown in Figure 2. Based on equations (1), (2) and (3) the measurement matrix is

$$H = \frac{p_1 - p_2}{\delta} \begin{bmatrix} K_{1,1} & K_{2,1} & \dots & K_{n+1,1} & \Lambda_1 & \Omega_1 \\ K_{1,2} & K_{2,2} & \dots & K_{n+1,2} & \Lambda_2 & \Omega_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ K_{1,M} & K_{2,M} & \dots & K_{n+1,M} & \Lambda_M & \Omega_M \end{bmatrix}$$

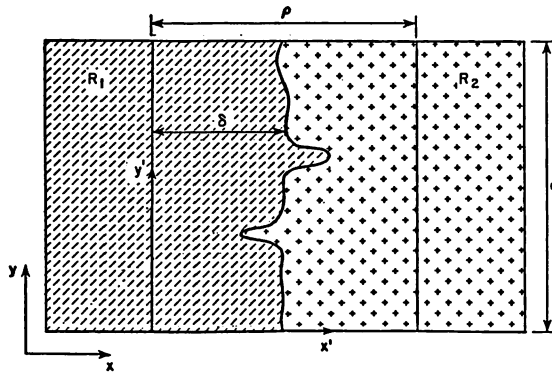


Figure 2. Boundary estimation nomenclature.

The success of the described scheme depends on two major factors: (1) choice of property p to be measured and (2) positioning of the knots. Various measurement schemes are described in [2]. (Results shown in this paper are generated by using mean measured along adjacent strips in regions R_1 , R_2 and ambiguity zone). In this paper we shall consider the role of knot positions.

The approximation to functions by splines is well known to depend on number and positions of the knots. Smooth curves, such as those considered in [1], can be adequately approximated by small number of equidistant knots. However, if abrupt changes are expected it is desirable to utilize large number of knots and it is particularly important that the knots be placed at the positions where significant changes occur. An approach to obtaining desired flexibility is to consider knots to be free parameters. Different schemes for handling variable knots have been described by Jupp [6] and De Boor [5]. Muth and Willsky [8] have designed a variable knot recursive scheme to approximate wave forms by splines. This method utilizes age-weighted linear recursive filter to change the locations of the knots. The scheme is designed to work on functions which have a large number of jumps.

In this work the assumption is that boundaries are generally smooth and abrupt changes occur infrequently. Therefore, it is more computationally efficient to achieve the flexibility of boundary approximation by knot additions rather than variations in the location of a given number of knots. The boundary estimation procedure starts with $(I+1)$ equidistant knots and the new knot is added to segment h_i at position r if $z_r - z_{r-1} > \Delta$ (Δ is the threshold value and z_r is the measurement taken along strip located at r). In this way new knots are added at the positions where abrupt changes in the first derivative occur or where maxima and minima are expected. The reliability of the measurement is increased by considering m previous and n new measurements around position r . While it is possible to consider large number of previous and new measurements, we have found $m=n=2$ to be sufficient. The dimensionality of the vectors (equations (1), (2), (3)) is increased by one for each additional knot and all vectors are appropriately

augmented. The final boundary is approximated by $N+1 \geq I+1$ knots. Addition of new knots is illustrated in Figure 3, while results obtained are shown in Figure 4.

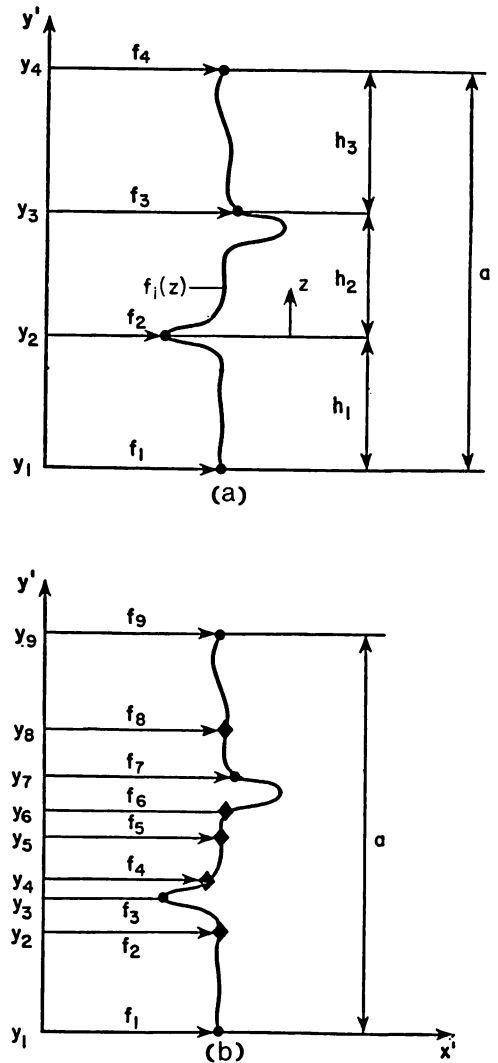


Figure 3. Addition of new knots (a) initial knots, (b) added knots (♦ represents added knots and • are initial knots).

4. CONCLUSIONS

Performance of the described boundary estimator has been evaluated on noisy images containing regions of uniform intensity as well as images containing texture. Generally, the results obtained are in good agreement with human perception. Addition of new knots, when required, allows good boundary approximation in cases where boundary location changes abruptly without adding significant computational burden. Development of the method is subject of further research. At present

we are investigating usage of multiple properties in the measurement scheme, i.e. both statistical and texture properties, to increase reliability of the measurements.

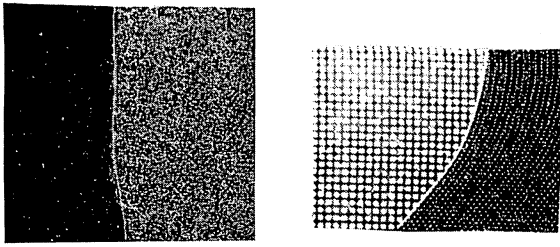


Figure 4. Examples of obtained boundaries in noisy images.

REFERENCES

- [1] Brzakovic, D. and Liakopoulos, A., "Estimation Theory Based Segmentation of Texture Images," in *Advances in Image Processing and Pattern Recognition*, V. Cappellini and R. Marconi (Eds), North Holland, Amsterdam, 1986.
- [2] Brzakovic, D. and Liakopoulos, A., "Measurement Models for Sequential Estimation of Boundaries in Digital Images," *Proc. of IASTED International Symposium on Applied Signal Processing and Digital Filtering*, Paris, France, 1985.
- [3] Burt, P. J. "Fast Transforms for Image Processing," *Computer Graphics and Image Processing*, Vol. 16, pp. 20-51, 1981.
- [4] Chen, P. C. and Pavlidis, T., "Image Segmentation as an Estimation Problem," *Computer Graphics and Image Processing*, Vol. 12, pp. 153-172, 1980.
- [5] DeBoor, C. A. *Practical Guide to Splines*, Springer-Verlag, New York, N.Y., 1978.
- [6] Jupp, D.L.B. "Approximation to Data by Splines with Free Knots," *SIAM Journal of Numerical Analysis*, Vol. 15, No. 2, pp. 328-373, 1978.
- [7] Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, Trans. ASME, Vol. D-82, pp. 35-45, 1960.
- [8] Muth, A.M.M. and Willsky, A. S., "A Sequential Method for Spline Approximation with Variable Knots," *International Journal of Systems Science*, Vol. 9, No. 9, pp. 1055-1067, 1978.
- [9] Otsu, N., "A Threshold Selection Method for Gray-level Histograms," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-9, pp. 62-66, 1979.
- [10] Pavlidis, T., "Structural Pattern Recognition," New York, Springer-Verlag, 1977.
- [11] Zucker, S. W., "Region Growing: Childhood and Adolescence," *Computer Graphics and Image Processing*, Vol. 5, pp. 382-399, 1976.

DETECTION OF SPECULARITIES IN COLOUR IMAGES USING LOCAL OPERATORS

Paul N.E. Pellicano
School of Computing Science
Simon Fraser University, Burnaby B.C.

ABSTRACT

Areas of spectral reflectance, or highlights, can be analyzed for a wide range of information or clues that they give us about a scene. This paper presents a local algorithm for analyzing a moderately unconstrained color image to determine the areas of spectral reflectance. The algorithm is based on the separability of the diffuse and spectral reflection components by differential methods.

The location of specular reflectances are marked by finding zero-crossings in concave down regions for two-dimensional arrays of intensities representing the color image. These zero-crossings correspond to the centers of the highlight regions. The highlight centers are then expanded to highlight regions by region growing in a direction orthogonal to the local orientation of the highlight. Thus, at the conclusion of the algorithm, the information known about each highlight includes location, size and direction.

INTRODUCTION

Computer Vision is often perceived as something that should be trivial. The reason for this perception is that we are ourselves so good at vision, we take the whole process of vision for granted. In fact, the interpretation of our three dimensional world, as portrayed in a two dimensional array of intensities, is anything but trivial. While humans bring a vast amount of 'intrinsic' information to bear on the problem of image analysis, the computer does not have the capacity, at the current time, to perform the same feat. Therefore, to permit any useful analysis of an image whatsoever, we tend to limit, or constrain our image world such that analysis becomes feasible with respect to the limited amount of knowledge we can impart to the computer.

A particularly useful and efficient task that we practice every day, however unwittingly, is that of distinguishing between objects made from different materials. An important prerequisite for such perception is the ability to discern the quality of an object's appearance. Various qualities of appearance are apparent in the world around us, such as texture, color, shine, luster, etc. all of which

give us important clues as to an object's composition. This paper will concern itself with the quality of surface gloss.

Glossiness, in general, is correlated with specular reflectance [Beck 72]. By looking at picture 1 we can easily determine which objects are shiny by the presence of areas of spectral reflectance.

Surface sheen, shine, gleam, etc. (see [Wyszecki 75] for a discussion of these terms) is a very important aspect in material discrimination. We regard metals as having a shiny appearance, whereas plastics, while they may have as smooth a finish, appear somewhat dull in comparison. Other surfaces may be altogether matte. These differences are caused by the presence, or absence, of local mirror-like or specular regions of reflected light, henceforth called highlights. If we can detect these highlights within an image, we can glean information that will help us to identify materials.

Some other consequences of finding highlights are that: (i) they would aid in constraining the size, color, and location of a light source; (ii) they would simplify object recognition or matching by identifying the regions so that some of the effects of the illumination could be 'factored out' and (iii) they also would enable constraints to be placed on object size and location [Thrift 82]. Perhaps a more basic or fundamental reason for wanting to locate highlights is that computer vision is concerned with modeling human vision, of which an inherent feature is the ability to locate highlights.

The detection of highlight regions proceeds by examining the stimuli that creates the sensation of a highlight. Horn's [Horn 75] model of surface reflectivity describes the two basic reflection components from a surface, specular and diffuse, as being separate quantities. Forbus [Forbus 77] used this information to generate a series of one dimensional profiles of intensity for curved surfaces, to see what parameters are relevant to the perception of highlights in archomantic images. Forbus noted that both the specular and diffuse reflection components must be present to create the sensation of a high light. Using this information, in conjunction with the expected sinusoidal shape of the specular reflection

component as given in Horn's equation, we can detect highlights.

1. DIFFERENTIAL OPERATORS

The sinusoidal shape of the specular component of reflection can be used to locate highlights by looking for zero-crossings in the first differential of the intensity image (see figure 1-1). Since zero-crossings in the first differential can correspond to either maxima or minima in the original image, care needs to be taken that we accept only those zero-crossings for which we have a maxima in the original image. Consequently, a concavity check is made by taking the second differential of the intensity image to make sure we have a peak of intensity and not a trough.

By using the first derivative of the Gaussian as our operator, we incorporate both a smoothing and differential operator into one step. The form of our operator is a two dimensional mask of values, calculated to simulate the derivative of the Gaussian, which can then be convolved with the intensity image. Since our convolution operator is now a function of σ , we can vary the sensitivity by using larger or small values for σ .

While we may choose to apply our differential operator using a single value for σ , that would be inappropriate considering the wide range of highlight sizes that may occur in an image. Highlights with a wide range of sizes can only be located by using multiple values for σ and then ORing all the results together. To ensure that we define zero-crossings not caused by singularities due to the choice of σ , the algorithm incorporates a phase which finds the zero-crossings for two values of σ and then ANDs those images together. Taking two values of σ relatively close together ensures that when we AND the results we do not eliminate 'true' highlights due to the scale of our operator. We can then OR a few such σ pairings to cover the gamut of highlight sizes.

However, we must remember that the differential mask is directional (non-isotropic) and must be applied to the image oriented at various angles of θ . It turns out that the convolution of the differential mask with the image need only be done twice, for two orthogonal angles, since the other angular orientations can be derived from those results. Complete zero-crossing angular orientation information can be ascertained from four angles of θ , each forty-five degrees apart. This means that two subsequent calculations need to be performed on the results of the orthogonal angle convolutions. Once we have the zero-crossing information for our differential masks the results can be combined with a concavity check to ensure we keep only those zero-crossings for peaks of intensity.

Concavity is determined by taking the second derivative with respect to a curve and looking at the sign of the resultant. In our case, this means convolving the original image with a two dimensional, second differential, mask. The mask is formed in the same manner as the first dif-

ferential directional mask, that is the differentiation is with respect to the Gaussian, but with the exception of using the Laplacian rather than the directional derivative so that the convolution need only be done once [Marr 75]. The result of the convolution is a two dimensional array composed of positive and negative regions demarcating concave up and concave down regions respectively. Only those zero-crossings within negative valued regions are accepted for further analysis. The result is an image containing zero-crossings for the original monochromatic image. There are still two other monochromatic images left from the original RGB images to process. Color is used to corroborate the data [Kanade 81] so that we have a 'true' highlight identification system. The complete zero-crossing information for the RGB image is shown in picture 2. Since zero-crossings chains are only one pixel wide, it is necessary to incorporate a region growing phase into the algorithm to locate the whole highlight.

2. REGION GROWING

The highlight pixels that surround the previously located highlight center chain can be found by growing outwards, from that central chain, in a locally orthogonal direction. To ascertain which direction is orthogonal the local direction of the highlight center chain is found over a three-by-three mask. The growing proceeds while the results of convolving a growing one dimensional orthogonal mask, with the original image, are increasing.

By convolving the power-of-two mask in figure 2-1 with the binary valued highlight chain image (1 for highlight pixels and 0 otherwise) and examining the resulting numeric value, we can determine the local direction. Using the eight-connected neighbor model there are four directions; north-south, east-west, southwest-northeast and northwest-southeast. For example if the convolution result is 17 or 68, the highlight center chain pixel is tagged with a east-west or north-south flag respectively. Similarly other values demarcate various compass directions. From the local direction labelling we can also label each of the pixels with a corresponding orthogonal direction. Choosing the local direction can be impossible for some patterns that can arise in a three-by-three area, so these pixels receive special treatment.

These pixels are labeled 'blob' pixels since they are without definite direction. To enable processing, they are marked with a flag that states every direction is orthogonal, and thus they are processed for each of the four directions. Once all the highlight center chain pixels have been tagged for direction, the orthogonal direction highlight growing can proceed.

Orthogonal to the center chain pixels which identifies the peak of the highlight the highlight intensity values decrease until reaching the value of the diffuse intensity for that surface. Therefore, a one dimensional mask, oriented in the orthogonal direction, is set up to calculate the difference between a highlight center chain pixel and points on each side of it (each side is treated separately

due to the non-symmetrical shape of most highlights). At each iteration of the algorithm the size of the mask is increased by one as the edge of the mask is moved 'outwards' by one pixel.

The algorithm keeps iterating for each highlight pixel point as long as the calculated value is increasing. When the algorithm terminates, the current value for the masksize determines the diameter of the highlight for the current side it was working on. Since blob pixels are processed for each direction, the current value of the mask size is stored so that it can be compared to the subsequent mask size values. The final mask size chosen for blob pixels will be the minimum of the directional mask sizes. The blob pixel will then be marked with one of the four directions that corresponds to the minimum sized mask (see picture 3 for a scene with the highlight diameters used to generate disks around the highlight center pixels). We now have each highlight chain pixel tagged with a highlight diameter to each side of it and a direction. The highlight is thus completely determined.

3. CONCLUSION

An algorithm is described in this paper which locally processes scenes of various objects to determine areas of spectral reflectance or highlights. The algorithm is based on the separability of the spectral from the diffuse reflectances by differential methods. Once the highlight center chains are detected by the algorithm, they are expanded to highlight regions by region growing in a direction orthogonal to the local orientation of the highlight. At the conclusion of the algorithm the information known about each highlight includes location, size and direction. Thus the algorithm provides information that a Computer Vision system must make use of when analyzing, or understanding, a scene. In addition, the information this algorithm provides can be used as a preprocessor to image processing algorithms that rely on predetermined areas of spectral reflectance. It can also be used to identify areas of spectral reflectance so that they can be removed from the scene. This eliminates illumination peculiarities which might confuse later/other algorithms that do pattern matching.

The fact that the algorithm is successful using moderately unconstrained images is important since it decreases the gap between the world that the computer can now understand and the extremely complicated one in which we live.

ACKNOWLEDGEMENTS

I would like to acknowledge the support and assistance of my supervisor, Dr. Brian Funt and the helpful criticisms of Kim Adamson-Sharpe. Nedenia Holm digitized the images contained in this paper and many others. This research was funded by Natural Sciences and Engineering Research Grant A4322.

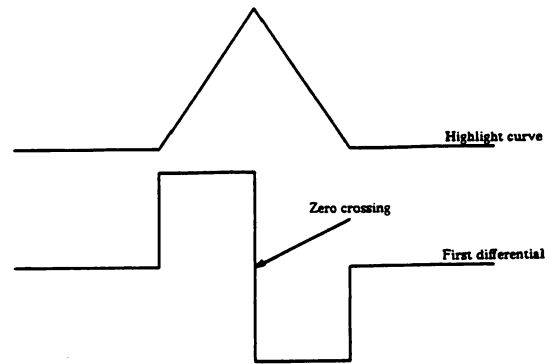
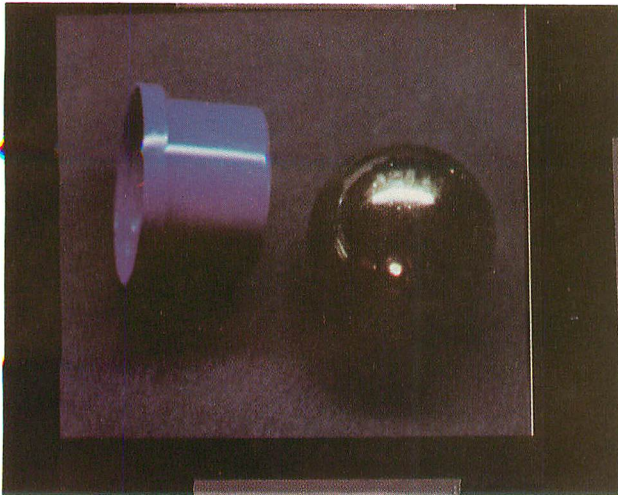


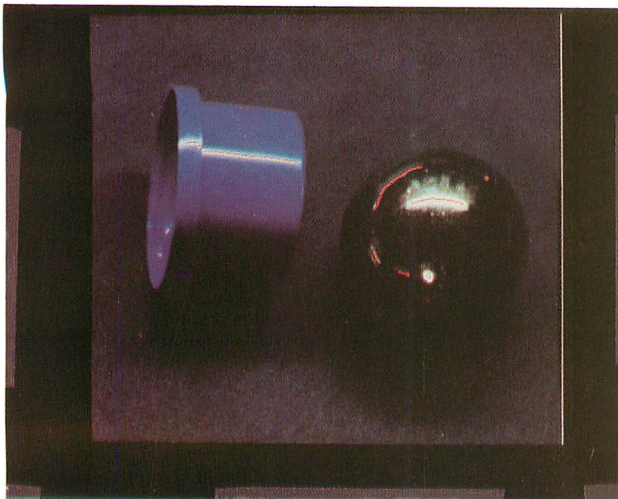
Figure 1-1: Highlight type curve and its derivative

2	4	8
1	0	16
128	64	32

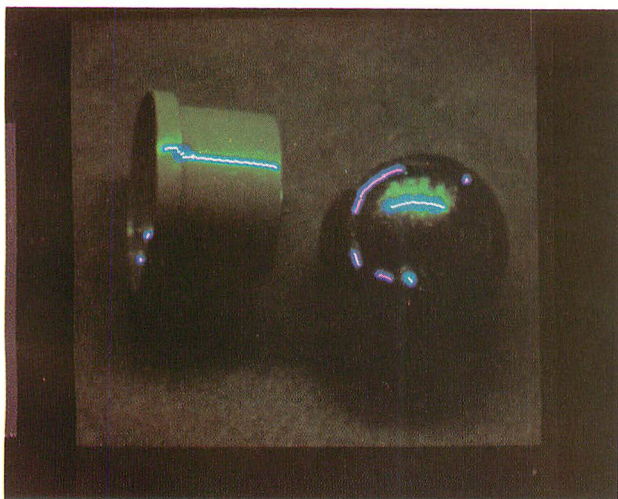
Figure 2-1: Power-of-two mask



Picture 1: Image containing objects with highlights



Picture 2: The highlight image (highlights in red)



Picture 3: Highlight diameters used to generate disks (disks are shown in blue)

REFERENCES

- [Babu 85] Babu, M.D.R., Lee, Chia-Hoang, Rosenfeld, A.
Determining Plane Orientation from Specular Reflectance.
Pattern Recognition (1), May, 1985.
- [Barrow 81] Barrow, H.G., Tenenbaum, J.M.
Artificial Intelligence 17
North Holland, 1981.
- [Beck 72] Beck, J.
Surface Color Perception
Cornell University Press, 1972.
- [Bryant 83a] Burt, P.J.
Computer Graphics and Image Processing 21
Academic Press, 1983.
- [Burt 83b] Burt, P.J.
The Laplacian Pyramid as a Compact Image Code
IEEE Transactions on Communications 3(4), April 1983.
- [Cook 82] Cook, R., Torrance, K.
A Reflectance Model for Computer Graphics.
ACM Transactions on Graphics 1(1):7-24, January, 1982.
- [Davies 81] Davies, E.R., Plummer, A.P.N.
Thinning Algorithms: A Critique and a New Methodology.
Pattern Recognition 14(1-6), 1981.
- [Forbus 77] Forbus, K.
Light Source Effects.
MIT AI Lab Memo (422), May, 1977.
- [Hall 83] Hall, R. Greenberg, D.
A Testbed for Realistic Image Synthesis
IEEE CG & A, November, 1983.
- [Horn 75] Horn, B.K.P.
Image Intensity Understanding.
MIT AL Lab Memo, August, 1975.
- [Kanade 81] Kanade, T.
Artificial Intelligence 17.
North Holland, 1981.

- [Marr 75] Marr, D.
Vision
W.H. Freeman and Company, 1982.
- [Pellicano 85] Pellicano, P.N.E.
Detection of Specularities in Color
Images Using Local Methods
Master's Thesis, Simon Fraser Uni-
versity, December, 1985.
- [Phong 75] Phong, B.T.
Illumination of Computer Generated
Images.
Commun. ACM, June, 1975.
- [Powers 79] Powers D.L.
Boundary Value Problems.
Academic Press, 1979.
- [Pratt 78] Pratt W.K.
Digital Image Processing
John Wiley and Sons, 1978.
- [Prewitt 70] Prewitt, J.M.S.
Object Enhancement and Extraction.
Picture Processing and Psychopic-
torics.
Academic Press, 1970.
- [Sears 58] Sears, F.W.
Optics
Addison-Wesley Publishing Com-
pany, 1958.
- [Shafer 84] Shafer, S.A.
Using Color to Separate Reflection
Components.
Carnegie-Mellon Technical Report
(136), April, 1984.
- [Thrift 82] Thrift, P., Chia-Hoang Lee.
Using Highlights to Constrain Object
Size and Location.
IEEE Publication, 1982
- [Ullman 75] Ullman, S.
On Visual Detection of Light
Sources.
MIT AI Lab Memo (333), May, 1975.
- [Wyszecki 75] Wyszecki, J.
Color in Business, Science and In-
dustry.
John Wiley and Sons, 1975.

COUPLING VISUAL AND DYNAMIC
FEATURES TO STUDY HANDWRITTEN SIGNATURES

Jean-Jules Brault et Réjean Plamondon

Laboratoire SCRIBENS
Département de Génie Electrique
Ecole Polytechnique de Montréal
Montréal, Québec H3C 3A7

RESUME

Un outil informatique est proposé dans le but de permettre l'étude interactive des caractéristiques spatio-temporelles des signatures manuscrites. L'outil permet de faire le pont entre l'aspect visuel d'une signature et les caractéristiques dynamiques reliées à son exécution. Des commandes d'édition graphiques et de traitement numérique sont disponibles à l'utilisateur pour respectivement manipuler et modifier à l'écran les différentes représentations reliées à une signature donnée.

ABSTRACT

A software tool is proposed for interactive study of spatio-temporal characteristics related to handwritten signatures. The tool fills the gap between visual and dynamic aspect of signature with specific graphic editing commands that can be used to manipulate on the screen the various representations of features related to a given signatures. Also, useful data processing commands allow modification of the content of these representations.

I- INTRODUCTION

Handwriting is a rather complex mechanism which results in the generation of line images. These images can be analysed by different recognition techniques based either on the visual output of the process¹ or on the dynamic information acquired by specific set up during the process itself. Among the different class of problems dealing with handwriting recognition, signature verification has been given a growing attention in the past ten years in the field of computer security. Indeed, with the increase in the number of electronic funds transfers and any other computer access, the need for an Automatic Personal Identification (API) system has become a major priority.

Signature verification techniques offer different² advantages over other identification techniques (passwords, PIN's, magnetic card, finger print, voice,...). It is an accepted and easily tested method. Signature cannot be lost or stolen and it can hardly be imitated dynamically. In the past fifteen years, intensive research has

been made^{3,4,5,6,7,8} to implant an API systems based on signature. But none of the systems already proposed in the literature has put the final point on the subject. Indeed, handwritten signature is a complex task requiring a high muscular skillfulness and we believe that further fundamental research is needed on the handwritten phenomenon to improve the performances of the systems.

Different tools and methods exist that help research in handwriting. In psychology⁹, for example, measurement of reaction time or movement time are often used to verify presumptions about specific handwritten task (e.g. identification of a movement unit in handwriting; the stroke). In this paper, we present an interactive software tool based on the possibility to do time coupling between visual information (that is the signature itself on the paper) and any type of dynamic information based on data sampled during the execution of the handwritten task.

In this contribution, we recall in section II the two class of features involved in signature verification (visual and dynamic) and point out their complementarities. We present in section III an overview of the features (technical and functional) implanted in the software tool. In section IV, we give a typical application showing the utility of the tool in interactive analysis of handwritten task, specifically signature.

II- VISUAL VERSUS DYNAMIC INFORMATION

Two major class of features dealing with signature can be used as input for an API system:

- the visual information
(that is the final result on the paper).
- the dynamic information
(that is the sequence in time of any measurable (but meaningful) characteristic).

Optical analysis of signature is a useful technique in off-line verification application like document expertise. But, sometimes, in order to pronounce a correct verdict about the authenticity of a signature, an expert needs to gather dynamic features from the static representation of a signature by examining it under microscope (the aspect of the paper fiber where the pen had passed, the variation in the thickness of the line,...). However these techniques cannot be automated (at least in a near future) and are thus unusable for API applications.

On the other hand, dynamic signals can be processed and analysed by different techniques, namely the usual signal processing approach (filtering, correlation, spectral analysis, time series, etc). But these methods rarely take into account the visual origin of these specific signals and questions like:

"Which parts of a signature was traced more rapidly than a speed threshold?"

"Are the amplitude variations of the signal more determining in the shaping of a given letter than are frequency variations?"

"Are rapid movements more accurate and easily repeated than slower ones?"

remain unanswered because only one aspect of the available information is usually analysed. We believe that signatures must be analysed by coupling the visual and dynamic information together. This requires that position and other dynamic information must be sampled simultaneously during the execution of a signature.

The source of information related to signature could be of various kind:

- Those issued directly from transducers at the execution time. The transducers could be in the writing pen (like strain gages⁵, accelerometers^{4,8}, etc.), or under the writing surface (digitizer⁷ or digitizer with analog computation¹⁰)

- Those obtained indirectly after computation from the sampled data (first and second time derivative of position, radius of curvature, instantaneous frequency of the signal, or any other interesting features).

But the main requirement to make time coupling is that all sources of dynamic information must be sampled simultaneously by an adequate set-up.

III- THE SOFTWARE TOOL

To be able to illustrate the idea behind the expression "time coupling", an handwritten signature has been sampled (at 200 Hz) by a digitizer (model MM960 Summagraphic Inc.) One can redraws the signature on a graphic screen by tracing the X(t) sequence vs the Y(t) sequence as in figure.1 where the bottom signal is X(t) while the one above is Y(t).

The effective coupling of information can be made internally by software because we know "when" each upstroke and downstroke have been made. To see a desired coupling with the software tool, the user only has to localize a moving cross on the screen with the cursor arrows on the keyboard (or with a mouse) on a point of interest (e.g. a glitch on the X(t) signal or a particular stroke on the signature) and depress the space bar on the keyboard (or a button on the mouse). The software automatically finds the point of the graph that is the nearest to the cross, and indicates with numbered vertical line(s) all the couplings in time occurring on the graphs that belong to the same signature.

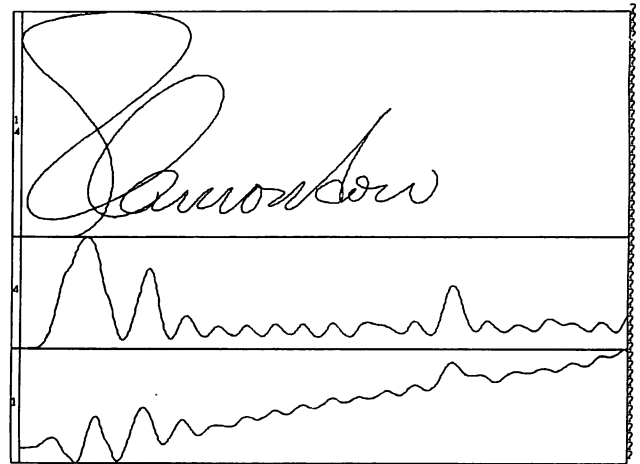


Figure 1

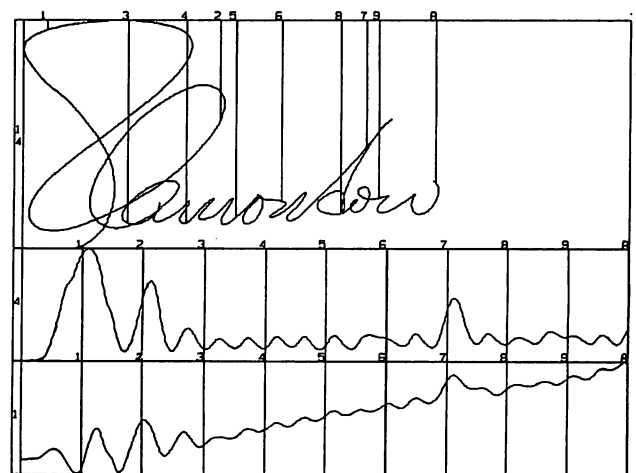


Figure 2

For example, the underlying "timing" of the signature in figure.1 can be retrieved by coupling ten consecutive samples of the position signal (e.g. one at each 100 samples representing one tenth of the total duration of the signature) to their effective positions in 2D (figure.2). The numbered vertical lines indicate the ten couplings.

Several functions have been implanted jointly to the coupling function but before saying more about that, the following describes the technical features of the software.

Technical features

The software is written in FORTRAN 66 (about 6000 statements or about 230 Kbytes of compiled code running on a mainframe IBM 4381). It uses the T.C.S. Library of subroutines and so it must be run from a TEKTRONIX 4010/4014 terminal (or a Tektronix emulator). At execution, an additional 170 kbytes is required for constants, and workspace.

At the data level, the working area consists in two regions (for eventually two signatures), each divided into seven channels. One channel can hold up to 2048 data (e.g. a ten seconds signature if sampled at 200 Hz). At start up, the program reads

a data file into the system and puts it in one region of the working area. If the file contains only three non-empty channels (e.g. X position, Y position, and pressure), four channels are then left empty and could be filled with signals that could be modified by one of the 19 data processing commands (see functional features).

At the graphical level, the software can trace on the same screen:

- up to 10 figures associated to the first file and/or
- up to 10 figures associated to the second file and/or
- up to 10 figures associated to the two files together in order to make comparison.
(e.g. correlation between files).

Each of these graphics can be modified in different ways by the 17 graphics commands as it is described next.

Functional features

The implanted functions are divided as follows:

- 19 data processing commands
- 17 graphic editing commands

As we have said, each channel of the working area can be modified by the data processing commands. These commands are menu driven. For example, one can filter, derive, interpolate or even make an FFT of a signals, or else, makes operations between signals (add, multiply, correlate...). Also one can add a customized application to the software. It is easily expandable.

Also, each figure drawn from these channels can be interactively edited via the graphic editing commands. These commands are called with a one character mnemonic and use a moving cross to manipulate the figures on the screen. Most of the graphics commands concern manipulations of two kinds of windows: a "virtual" window which determines the data to be displayed, and a "display" window which determines where, on the screen, the selected data will be plotted. Commands are available to modify the number, the disposition and the content of the windows to be drawn on the screen. It is also possible to make measurement by superimposing a user defined reference grid on a window or an other window in order to make qualitative comparison.

Obviously, we cannot discussed here the details of each of the implanted functions but we can give an application example using some commands in order to gained an insight in the way the program progress throughout the commands.

IV- APPLICATION

Suppose we have a signature sampled with a given digitizer. We want to know where, on the signature, the line has been traced at a speed higher than an arbitrary threshold value.

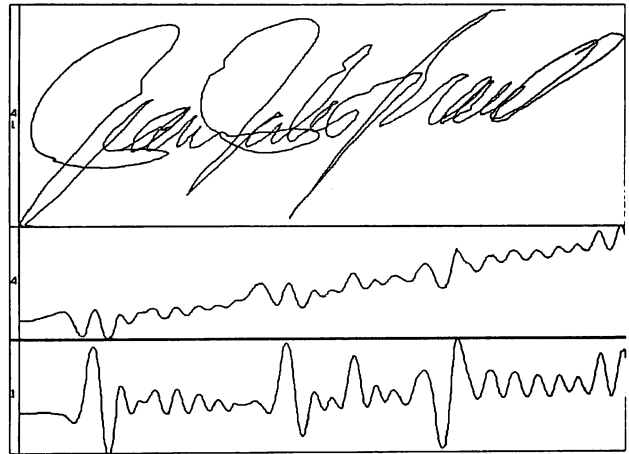


Figure 3

To answer to this question, the following procedure will be used:

1. Read the file where the X and Y coordinates of the signature are stored.
2. Make a copy of the position signals $X(t), Y(t)$ in two empty channels of the working area.
3. Display the signature to see if it is necessary to filter the signals (figure.3).
4. Filter if necessary (yes in our case).

N.B.: Two kinds of filter are available in the program:

- frequency sampling FIR filter¹¹
- moving weighted average filter.

5. Derive one pair of position signals to obtain the speed relative to each direction $X'(t)$ and $Y(t)$.

N.B.: We use finite central difference calculus for fifth orders polynomials¹².

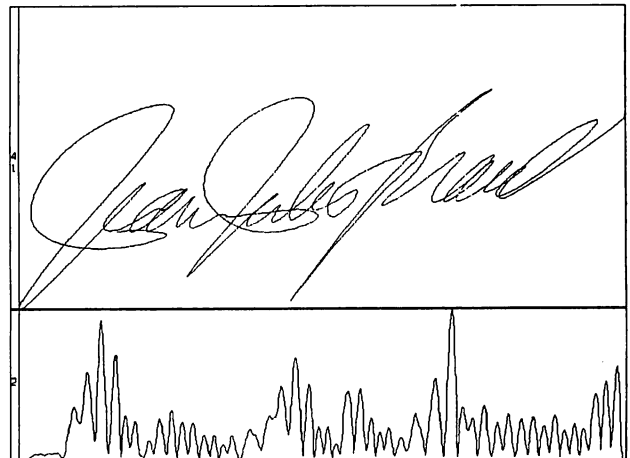


Figure 4

6. Transform the $(X'(t), Y'(t))$ signals into $|V(t)|$, the absolute speed value.
7. Display the signature and the $|V(t)|$ signal (figure.4).
8. Choose the threshold speed value (below which the signature will be traced with dotted line) with the specific graphic editing command.
9. Display again the signature and $|V(t)|$ (figure.5)

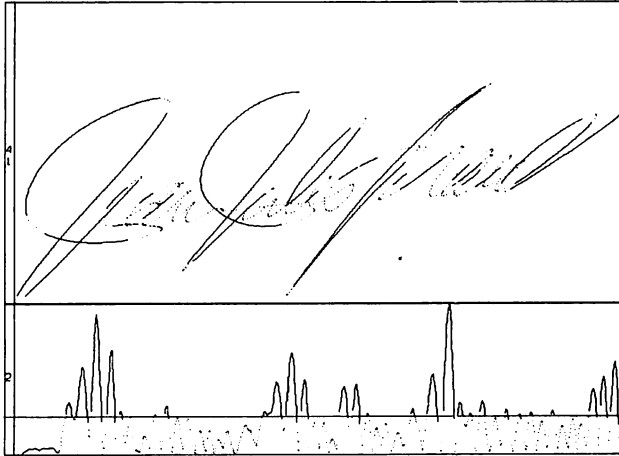


Figure 5

If we now want to "zoom" on a particular section of the whole signature, a command exist to cut the desired section and redraw it magnified (e.g. the "an" of "Jean"). Figure.6 shows the result.

As we have already said, other useful functions are available to help the user in his study of a handwritten task and figure.7 shows some of them. A more complete subset of functions will be shown at the oral presentation.

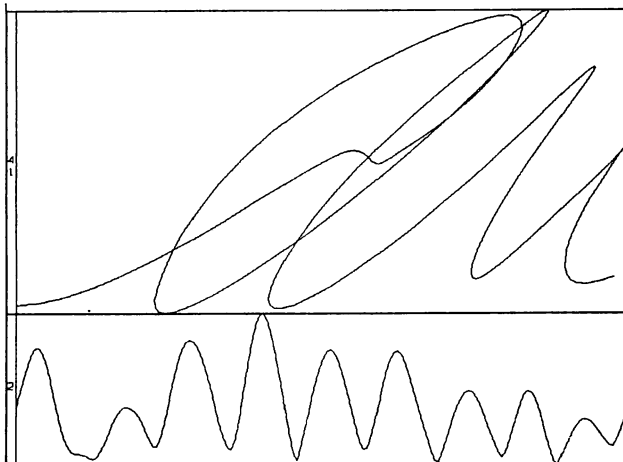


Figure 6

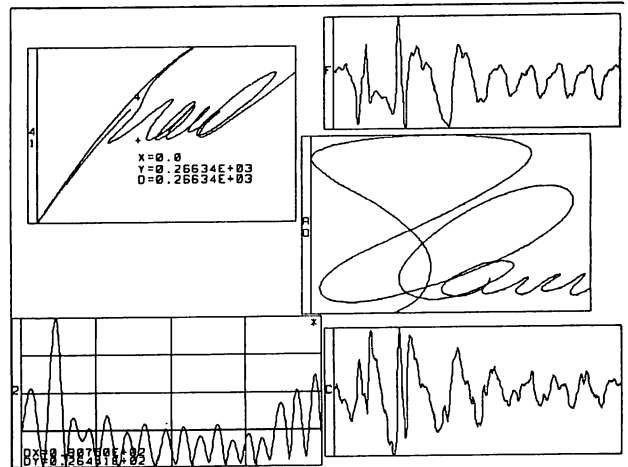


Figure 7

V- CONCLUSIONS

We have developed a tool of practical interest to study signatures (or handwriting in general) in a pleasant environment. It runs on a mainframe allowing computing power and a large library of already developed programs of mathematical transformations and digital signal processing. The program has provision for additional functions. The features extraction process can now be made interactively with this software tool. We presently use it to develop handwriting formation models (and to verify model already proposed in literature⁹) in order to better understand the nature of handwriting.

VI- BIBLIOGRAPHY

1. Suen, C.Y., Berthod, M. and Mori, S., "Automatic Recognition of Handprinted Characters - The State of the Art", Proc. of the IEEE, Vol.68, no.4, 1980.
2. Long Range Planning Service Service, "Automated Personal Identification", Stanford Research Institute, 1974.
3. Liu, C.N. and Herbst, N.M., "Automatic Signature Verification Based on Accelerometry", IBM Journal of Research and Development, Vol.21, pp.245-253, 1977.
4. Brault, J.J., "Design d'un Crayon Accélérométrique pour la Vérification Automatique des Signatures" Mémoire de Maîtrise, Ecole Polytechnique de Montréal, janvier 1983.
5. Crane, H.N. and Ostrem, J.S., "Automatic Signature Verification Using a Three-axis Force-Sensitive pen", IEEE Trans. Systems Man, and Cybernetics, vol.SMC-13, no.3, pp. 329-337, 1983.
6. Brault, J.J., Plamondon, R. "Histogram Classifier for Characterization of Handwritten Signature Dynamic", Proc. 7th Conference on Pattern Recognition, p.619-622, Montreal, 1984.

7. Lorette, G., "On Line Handwritten Signature Recognition Based on Data Analysis and Clustering", Proc. 7th Conference on Pattern Recognition, p.1284-1287, Montreal, 1984.
8. Worthington, T.K., Chainer, T.J., Williford, J.D., Gundersen, S.C., "IBM Dynamic Signature Verification", Computer Security, North-Holland, IFIP, 1985.
9. Acta Psychologica: Motor Aspects of Handwriting, International Journal of Psychonomics, North-Holland, Amsterdam, vol.54, 1983.
10. Maamari, F. and Plamondon, R., "Extraction of the Analog Pen Tip Position, Velocity and Acceleration Signals from a Digitizer", Neuronal and Motor Aspects of Handwriting, North-Holland Publishers, Editor Dr. H.S.R.Kao, 1985.
11. Rabiner, L., Gold, B., McGonegal, C.A. "An Approach to the Approximation Problem for Nonrecursive Digital Filters", IEEE Transactions on Audio and Electroacoustics, Vol-18, No.2, June 1970.
12. Hornbeck, R.N. "Numerical Methods", Prentice-Hall Inc, New Jersey 1975.

DETECTING GLASS FIBERS USING COMPUTER VISION

Alfred S. Malowany Martin D. Levine Mussa R. Kamal[†]
Ronald Kurz Abdol-Reza Mansouri

Computer Vision and Robotics Laboratory
Department of Electrical Engineering
McGill University
Montréal, Québec, Canada

[†] Department of Chemical Engineering, McGill University, Montréal, Québec, Canada.

Abstract

We present an application of computer vision in the field of chemical engineering, the processing of images of glass fibers. Fibers are used as reinforcement for several polymer products to enhance their mechanical and thermal performance. In order to evaluate the properties of the polymer products however, a quantitative measure of fiber length and orientation is required.

The algorithm presented here locates the fibers present in a given image and thus enables their quantification in length and orientation. This algorithm operates in two steps. In the first step, feature points in the image are extracted in order to enable the generation of hypotheses as to the possible presence of fibers. In the second, the generated hypotheses are verified and the hypotheses that yield the highest confidence are retained.

1. Introduction

We present an application of computer vision in the field of chemical engineering, the processing of images of glass fibers. Fibers are used as reinforcement for several polymer products to enhance their mechanical properties and thermal performance. However, the process induced orientation distribution of fibers is anisotropic, and this results in direction dependent mechanical properties. Thus, a quantitative measure of fiber length and orientation is required for predicting the mechanical properties of the product.

In a typical image of glass fibers, the fibers have a high intensity, as opposed to a noisy background which has, on average, a low intensity. In addition, the fibers appear as straight line segments of differing lengths. Thus, looking for fibers, we actually focus our attention on detecting high intensity straight line segments on a low intensity and noisy background. A number of techniques have been developed in order to solve this problem. Relaxation labelling is one of them[1]: using this technique, an initial set of orientations is assigned to the image points, and these orientations are iteratively altered until the orientation at each point becomes the one dictated

by its neighborhood. The main drawback of relaxation labelling and similar techniques lies in their excessively large computational complexity, which renders them impractical for most applications. Another technique that is in use for detecting line segments is the Hough transform[2]. The Hough transform is a mapping from image space to parameter space (usually taken as distance and orientation) in which colinear points in the image appear as clusters of points in the parameter space. However, the Hough transform does not distinguish between connected and non-connected points. This results not only in interference between points residing on different segments, but also in ambiguous interpretations of the clusters in parameter space.

We take a different approach for solving the fiber detection problem. For this, we adopt the well known hypothesis prediction/verification paradigm [3]. This algorithm consists of two major parts: a predictor, which predicts possible fiber locations, and a verifier, which verifies the predictions and discards those which do not satisfy all of the constraints. Bearing in mind that template matching is itself a form of hypothesis prediction/verification (although of an exhaustive nature), it becomes clear that the main requirement that is imposed on the predictor is that it should significantly limit the scope of its predictions. In other words, the image-based features that are used by the predictor in formulating hypotheses should be closely correlated with the hypotheses themselves. In what follows, the features that are used in formulating hypotheses are described, and the cost function which allows the verifier to prune hypotheses is presented.

The complete algorithm is implemented in the C programming language on a VAX 11-750 running the UNIX operating system.

2. Predicting Fiber Locations

Consider a slide of glass fibers. The slide is illuminated by direct lighting and the reflected light pattern is viewed with a camera mounted on a microscope. This yields an image in which the regions occupied by the fibers have a large gray value, as opposed to the background gray value, which occupies, on average, the low

intensity range. Thus, in such images, the histogram is bimodal, and it is possible to differentiate between the fibers and the background by simple intensity thresholding. The threshold is selected using the p-tile method, whereby a certain percentage of the pixels are assigned to the background, while the remaining pixels are assigned high intensities and are identified as forming the regions occupied by the fibers.

In our experiments, the threshold is chosen in such a way as to assign 85 percent of the pixels to the background, and only 15 percent to the fibers. This specification is directly related to the density of the fibers. Since this density is relatively constant, the percentage value specification is invariant with respect to the type of images that is being dealt with. As a result of intensity thresholding, the image is partitioned into a set of regions, all of which (except for the background) have resulted either from the glass fibers or from noise.

In order to partially eliminate noise, regions with very small areas are discarded. In order to detect the fibers, even those inside large clumps, and then measure their length, the geometrical properties of the remaining regions have to be analyzed. These geometrical properties are captured by first thinning the regions. The thinning algorithm used is Tsuruoka's sequential algorithm for 4-connected thinning of binary images[4]. The advantage of thinning lies in the fact that it reduces the amount of data to be processed at later stages, while preserving its integrity as far as the number of fibers, as well as their individual length and orientation are concerned. In addition, thinned regions allow us to formulate hypotheses as to the placement of the fibers. The observation underlying this statement is that thinning imposes a structure on the local arrangement of pixels inside a region. Since our objective is to detect glass fibers, which are straight line segments, knowledge of the endpoints of a glass fiber is sufficient to determine its position, length and orientation. Thus, in order to produce hypotheses as to the placement of the fibers, we will, in the first step, detect those pixels which could be located either at the extremities of fibers, or at their intersection. Knowledge of these endpoints and intersection points will allow us to predict all possible configurations of fiber arrangements, and then retain only the one configuration that best matches the fiber arrangement in the original image.

In order for a pixel **P** to be an end-point, it should possess the following neighborhood configurations (modulo reflections and rotations):

$$\begin{pmatrix} \times & \mathbf{P} & \times \\ \times & \times & \times \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} \times & \mathbf{P} & \times \\ \times & \times & \times \end{pmatrix}$$

where the \times sign denotes a pixel which belongs to the same region as **P**. Also, in order for **P** to be an intersection point, it has to have the following neighborhood configurations (modulo reflections and rotations):

$$\begin{pmatrix} \times & \times \\ \mathbf{P} & \times \\ \times & \times \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} \times & \times \\ \times & \mathbf{P} & \times \\ \times & \times \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} \times & \times \\ \times & \mathbf{P} & \times \\ \times & \times \end{pmatrix}$$

Let **R** be a thinned region, defined by its constituent image points. Let **S** be the set of end-points and intersection points which also belong to **R**. Since a line segment can be defined by a pair of points, generating hypotheses as to the presence of fibers is equivalent to mapping the set **S** into the Cartesian product of **S** with itself, i.e. **S** \times **S**. The elements of **S** \times **S** are of the form (P_i, P_j) where P_i and P_j are end points or intersection points of the thinned region **R**. Note that the hypotheses which correspond to lines of length 0, i.e. to elements in **S** \times **S** which are of the form (P_i, P_i) are not considered. In addition, since for our purposes the line segment predicted by the pair (P_i, P_j) is the same as that predicted by the pair (P_j, P_i) , we need consider, in total, less than half of the generated hypotheses. Once all possible hypotheses have been generated, they are forwarded to the hypothesis verifier which retains only those which could be valid, according to some specific criterion.

3. Verifying Hypotheses

Let **H** be the set of all possible hypotheses. We have **H** = **S** \times **S**. As was mentioned previously, we consider only those hypotheses which are neither trivial nor redundant. Evaluating the generated hypotheses and retaining those which could be true is equivalent to mapping **H** into a set of hypotheses **H***, where all hypotheses belonging to **H*** are true according to some specific criterion. This criterion should reflect as much as possible the information in the original image **I**(x, y), as far as the placement of fibers is concerned. One criterion that could be used in evaluating the hypothesis **h** \in **H** would be the following:

$$\mathbf{F}(\mathbf{h}) = \frac{1}{N} \sum_{(x,y) \in L_h} \mathbf{I}(x,y)$$

where L_h is the line segment predicted by hypothesis **h**, and N is the number of points on L_h . Since the background has, on average, a lower intensity than the regions occupied by the glass fibers, the above summation will yield a large value whenever the line segment corresponding to a predicted hypothesis actually overlaps with a glass fiber, and a much lower value if the overlap is only partial. We would thus have:

$$\mathbf{h} \in \mathbf{H}^* \equiv \mathbf{F}(\mathbf{h}) \geq T$$

where T is some threshold imposed on the criterion **F**. This approach could work in some limited cases. Its main drawback is its dependence on variations in the input image, namely, in the gray levels of the regions occupied by the fibers. In order to remove this dependence, the predicted hypotheses are not matched with the original

image, but rather with the thresholded image, where regions occupied by the fibers have a gray value of 1 and the background has a gray value of 0. Thus, our new criterion is:

$$F(h) = \frac{1}{N} \sum_{(x,y) \in L_h} I_T(x,y)$$

where I_T is the thresholded image. The above criterion returns a value between 0 and 1. Again, as before, a hypothesis h is retained if and only if its associated criterion function $F(h)$ has a value larger than a threshold T . The closer this threshold is to 1, the better the match is between the selected hypotheses and the actual fiber placements. Conversely, if T is chosen close to 0, a large number of false hypotheses will be retained. Thus, the tradeoff on the choice of the threshold T is the traditional tradeoff between accepting a hypothesis given that it is false, and rejecting it given that it is true. Note that we could also have matched the predicted hypotheses with the thinned image. The main drawback with doing this however, lies in the lack of robustness of such a match to slight variations in line orientations (which would be due to noise), bearing in mind that line segments in the thinned image are only one pixel wide, while they are many pixels wide in the thresholded image.

Once the set H^* of fiber locations has been generated, additional processing is necessary in order to group together line segments which have been artificially split. This happens whenever many fibers intersect. The result of each intersection is an additional intersection point, which yields additional hypotheses. In order to overcome this, the cosine of the angle between any two segments in H^* which have a point in common is used as a similarity measure to decide whether or not two segments are actually part of a common larger segment.

A summary of the complete algorithm is shown in figure 1.

4. Experimental Results

Figure 2 shows a typical image of a slide of glass fibers. Note that in addition to the clusters formed by the fibers, complications are introduced by the presence of background noise.

Figure 3 shows I_T , the result of thresholding.

Most of the fibers have been retained during the thresholding process owing to their large gray value; however a number of regions which are due to noise have also been detected. These regions, which are small in size, are eliminated in a subsequent step. Figure 4 shows the result of thinning the thresholded image.

As expected, the amount of data to be processed has been drastically reduced, while shape information relating to fiber placement and orientation has been preserved. Figure 5 shows the result of the hypothesis prediction/verification process. The threshold T that is imposed on the quality of the match is chosen equal to 0.9.

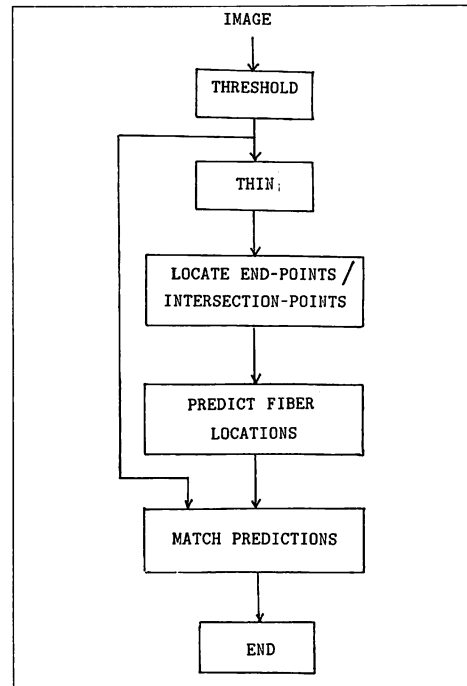


Figure 1 Flowchart of the algorithm

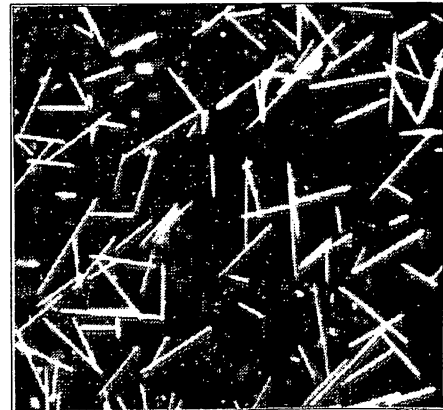


Figure 2 Original image

In other words, in order for a certain hypothesis to be verified as true, there must be at least 90 percent of the line predicted by that same hypothesis overlapping the region occupied by the fibers in the thresholded image.

As can be seen, a large number (80 to 85 percent) of fibers have been properly detected. Fibers which have not been detected are mostly associated with noisy regions, or even regions with an excessively large number of intersecting fibers, where the shape information is distorted during the thinning process. This distortion results in a series of false hypotheses, which are then rejected during the hypothesis verification process, owing to the rather poor match between the predicted line segments and the actual fiber locations. Conversely, a number of line seg-



Figure 3 Thresholded image

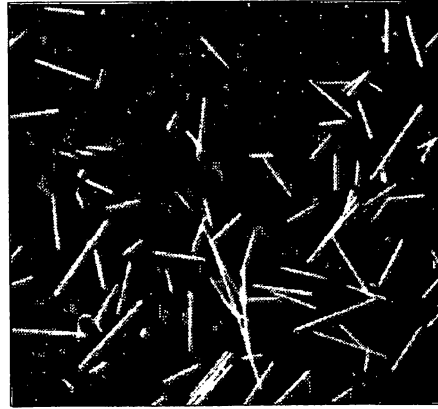


Figure 6 Original image

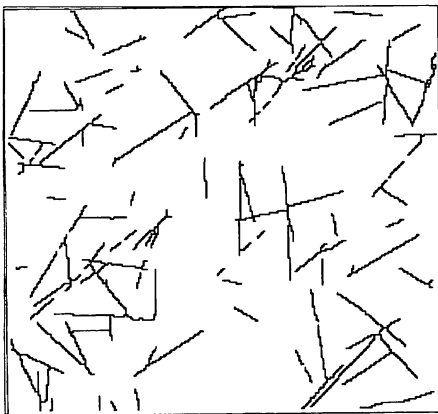


Figure 4 Thinned image

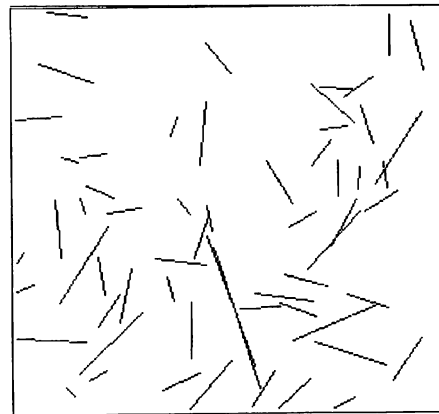


Figure 7 Detected fibers



Figure 5 Detected fibers

ments which are not associated with fibers but rather with noisy clusters are also detected.

Figure 6 shows another image of glass fibers.

In figure 7, the result of the processing is illustrated. Again, the same proportion of glass fibers has been detected.

The procedure currently distinguishes between 75 and 95 percent of fibers present in a given image, depending mainly on its complexity. The use of computer vision for glass fiber detection has yielded a dramatic increase in the analysis speed of the cross-sectioned experimental samples.

5. Conclusion

In this paper, an efficient algorithm for detecting glass fibers in polymer products was presented. The algorithm consisted of two parts: a predictor, which predicted possible fiber locations, and a verifier, which matched the predictions against information derived from the original image. The results of applying this algorithm to images of glass fibers were shown to be successful.

Although this algorithm currently distinguishes between 75 and 95 percent of the fibers present in a given image, possible improvements can be made by refining the segmentation procedure used in this algorithm; this will lead to a better discrimination between fibers and background, and will hence reduce the loss of shape information.

6. References

1. S. W. Zucker, R. A. Hummel and A. Rosenfeld, "An Application of Relaxation Labelling to Line and Curve Enhancement," *IEEE Transactions on Computers*, Vol. 26, No. 4, April 1977, pp. 394-403.
2. R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM*, Vol. 15, January 1972, pp. 11-15.
3. A.-R. Mansouri, A. S. Malowany and M. D. Levine, "Line Detection in Digital Pictures: A Hypothesis Prediction/Verification Paradigm," Technical Report 85-17R, Computer Vision and Robotics Laboratory, Department of Electrical Engineering, McGill University, Montreal, Quebec, Canada, August 1985.
4. Spider User's Manual, Joint System Development Corp., pp. III-546, III-547, 1983.

Acknowledgements

This work was partially supported by the Natural Sciences and Engineering Research Council and the Ministry of Education of the Province of Québec. Martin D. Levine would like to thank the Canadian Institute for Advanced Research for its support.

RECONSTRUCTION AND DISPLAY OF THE RETINA

Kenneth R. Sloan, Jr.

David Meyers

Department of Computer Science, FR-35

University of Washington

Seattle WA 98195

U.S.A.

Christine A. Curcio

Departments of Biological Structure and Ophthalmology

University of Washington

Seattle WA 98195

U.S.A.

ABSTRACT

The retina is an approximately spherical structure. In order to gather information such as the density of rods and cones it is necessary to flatten the retina. It is desirable to project these measurements back onto the original spherical form of the retina, interpolate the sampled data, and display the results. This paper is a summary of techniques which we have developed to perform these tasks.

RÉSUMÉ

La rétine peut être approximée par une surface sphérique. Pour recueillir certaines informations comme la densité des cônes et des bâtonnets, il faut aplatir la rétine. Nous aimerions reprojeter ces mesures sur la surface sphérique originale, interpoler les données recueillies et afficher les résultats. Cet ouvrage est une synthèse des techniques que nous avons développées pour effectuer ces tâches.

KEYWORDS: human retina, reconstruction, spherical geometry, interpolation.

1. Introduction

The topography of the constituent cells and efferent pathways of the retina is important for understanding how the visual world is sampled and how it is represented in the central nervous system. The retinal whole mount is the histological method of choice for revealing these topographical relationships (Stone, 1981, for review). Because the retina covers the major part of the sphere, it must be cut so that it can be flattened for viewing under a microscope. Thus, a general problem with whole mounts is that spatial relationships are lost across the cut edges. Furthermore, locations of features on the retinal sphere, which in theory could be specified with great precision, are not readily determined from their positions in the flattened tissue.

These problems may be solved by reconstructing the original spherical surface from the flattened tissue. Such a reconstruction has been accomplished manually by approximating tracings of the tissue to the surface of a sphere of appropriate diameter (*e.g.*, Østerberg, 1935).

The advent of sophisticated and affordable computer technology has made digital reconstruction techniques possible. We report methods for specifying a retinal coordinate system, reconstituting the retinal sphere from a three-piece whole mount (Curcio, *et al.*, in preparation), and displaying topographic data.

Our key reconstruction step relies on the fact that one of the three pieces of the retina has a particularly easy mapping back to the sphere, based on natural landmarks. Once this piece has been placed on the sphere, the other two are positioned relative to it, using a small set of fiducial points. In the first case, we can assume that there has been little or no distortion of the tissue. For the second placement problem, we cannot make this assumption. Instead, we assume that the tissue has been warped, and rely on an iterative relaxation procedure to place each point.

The reconstructed retina is then used as the basis for display. We construct a triangular mesh connecting

This work was supported in part by the National Science Foundation under grant number DCR - 8505713, by a Lions Northwest Training Fellowship, and by the National Institutes of Health under grant number EY04536.

the sampled points. Measured quantities (rod, cone and ganglion cell densities) are represented as intensities (usually false-colored). The triangular mesh can be directly displayed to give a direct three-dimensional view of the retina, or projected onto the plane in the style of conventional visual field maps. To generate a true visual field map, we can back-project the retina through a standard model of the eye's optics.

The display techniques have been applied to Østerberg's data (1935), and have provided invaluable guidance in the design of the sampling scheme which we are using now.

2. Coordinate Systems

2.1 Retinal Sphere

The retina is treated as the surface of a unit sphere. Any point on the sphere can be indexed by two coordinates, λ (longitude, meridian) and ϕ (colatitude, eccentricity). This coordinate system allows us to make comparisons between eyes of different diameters.

Eccentricity is measured from the center of the fovea. The nasal horizontal (0°) meridian is defined as the meridian passing through the center of the fovea and the center of the optic disk. The superior vertical meridian is at 90° , temporal horizontal meridian is at 180° , and the inferior vertical meridian is at 270° .

2.2 Microscope stage

We developed a 3-piece whole mount dissection technique which results in a belt 60° wide roughly centered on the horizontal meridian, and two caps from the inferior and superior retina. These three pieces can be flattened without tearing the retina; the belt is only very slightly distorted.

The locations of data points on the whole mount are expressed in terms of the X,Y coordinates of the microscope vernier. The raw data base consists of a collection of such X,Y points, along with measurements made at these locations, such as the density of rods or cones.

In order to assist in the reconstruction, we also note the positions of the fovea, the optic disc, and several (about ten) *key points* (usually blood vessels) along each of the shared boundaries between the belt and the caps.

2.3 Visual field

It is important to be able to express retinal location in terms of functionally defined locations in the visual field. The projection of the visual field onto the retina has been deduced by tracing the path of rays through the optical apparatus of an average eye. The projection is nonlinear, such that a degree of visual angle subtends a greater extent of retina centrally than peripherally. The exact nature

of the nonlinearity varies among different schematic eyes, based on different underlying assumptions (e.g., Drasdo and Fowler, 1974). One advantage of our decision to keep all retina data in a retina-based coordinate system is that it remains available for transformation to visual field coordinates by any suitable schematic eye.

3. Mappings

3.1 Three Planar Patches \Rightarrow One

The three separate planar patches (Belt, Inferior Cap and Superior Cap) are positioned in a common, planar coordinate system so that (see Figure 1):

- the fovea is at the origin
- the optic disc is on the positive x axis
- the inferior and superior caps are attached to, and tangent to, the belt, at a common key point.

3.2 Plane \Rightarrow Sphere

Mapping the central belt to the sphere is relatively easy. The shape and extent of this patch was chosen so that it could be flattened easily, without appreciable distortion. The fovea and optic disc provide all the landmarks necessary to orient the planar stage coordinates with respect to our spherical coordinate system. All that needs to be done is to wrap the rectangular belt around the sphere. First, we estimate the diameter of the retinal sphere by measuring the outer diameter of the eye, and the thickness of the sclera. The $\langle x, y \rangle$ coordinates can then be scaled from mm (as measured by the microscope vernier) to degrees of arclength (as measured on the unit sphere). They can then be considered to be two sides of a right spherical triangle. The $\langle \lambda, \phi \rangle$ coordinates are straightforward to calculate from this triangle.

This calculation correctly maps all points in the belt (including the belt's version of the keypoints) onto the sphere. However, it provides only a very gross estimate of the position of a point in either of the caps.

In order to position a cap point, we use an iterative relaxation process. In the planar stage coordinates, we calculate the distance between the cap point and each of the key points in that cap. After mapping to the sphere, we discard the key points associated with the cap, and look at the corresponding key points in the belt. These points have been correctly positioned on the sphere. We measure the (great circle) directions and distances between the cap point and each key point. Of course, the distances will be different than those measured in the plane. By taking the vector sum of these differences, and moving the cap point (on the sphere) until this sum approaches zero, we find a point on the sphere which minimizes our placement error. The process is observed to converge in some 10-30 iterations.

It is very important to properly weight the effect of individual key points. On the one hand, each key point provides some information about the proper placement of a particular point. On the other hand, key points which are very far away are less reliable, primarily because the warping is likely to be very non-linear over large distances.

Giving each key point equal weight results in obvious, gross errors. Weighting each key point by the inverse of the (planar) distance to the point to be placed is significantly better, but still produces an occasional misplacement. Inverse square distance is currently in use, and appears to appropriately balance the contribution of each key point to the final placement.

After this mapping, we have the original data points positioned in our canonical spherical coordinate system. See Figure 2.

4. Triangulation

Effective display of this data requires more than isolated data points. We would like to fit a surface to these points, and use that surface to provide estimates of the measured quantities everywhere on the retina. A first step in this direction is to tessellate the sphere with triangular patches, using the data points as vertices.

The data points are connected into a triangular net by projecting them back into the plane and computing local equiangular triangulation of the projected points, which is the completion of the Delaunay tessellation (Sibson, 1978). Given the relatively small size of our data (typically 200 points), we directly calculate the equiangular triangulation.

The projection to the plane used is the equidistant (polar azimuthal equidistant) projection (Frisen, 1970) - achieved by treating the spherical (λ, ϕ) coordinates as *polar* coordinates. This projection preserves radial distances (eccentricity), while stretching tangential distances. The surface of the sphere maps into a disc of radius π . The worst distortion is at the opposite pole, which maps to the circle surrounding this disc.

In the plane, the Delaunay triangulation is optimal in the sense that the triangles are as compact as possible. When projected back to the sphere, this triangulation is very good near the fovea, and less so near the periphery.

5. Display

The last problem is to display this spherical triangular mesh, with measured density values at each vertex. The customary method of presenting visual field data is to project to the plane (using the equidistant projection) and plot the density values as gray levels (or isodensity contours). It is easy to augment this display with gamma correction and false coloring (Sloan and Brown, 1979). We have a choice of painting each triangle with an average intensity value, or of interpolating the values measured at the vertices (see Figures 3-7). In either case, the triangulation developed above is exactly right for this projection.

This display is similar to those that anatomists and ophthalmologists are used to seeing, and have little trouble interpreting. We can also, of course, display the triangular mesh (colored as above) as a three dimensional, spherical surface. Going further, we can use the measured densities to deform the surface away from the sphere. Finally, we are beginning work on fitting a smooth surface to the data, using our triangular mesh as a control graph. The methods of (Farin, 1983) can be applied directly to our data.

6. Discussion

Østerberg's paper (1935) on the distribution of rods and cones in the human eye is one of the most widely cited studies in the vision literature. The findings most frequently cited and reprinted are the density of rods and cones along the horizontal meridian and the calculation of the total number of photoreceptors. His data on the overall topography of photoreceptors, illustrated by density maps (contours for rods, symbols for cones) are less well appreciated, mainly because of the relatively less accessible nature of his maps. Our display techniques, applied to this data, provide much better intuition about the gross topography, and have pointed out deficiencies in his sampling scheme.

For example, look at Figure 6, which shows the central 8° of Østerberg's eye. Notice that he sampled very finely along the 0° meridian, but much more coarsely in other directions. When this sampling is displayed directly, it gives a distorted picture of the shape of the density map. If one assumes that the map is radially symmetric, then a better-looking density map could be constructed (by duplicating the 0° points), but this "better-looking" picture would be more a product of the assumption than of the measured data. By comparison, look at Figure 7, which shows the same central 8° of a more recent eye.

7. References

1. Curcio, C.A., D. Meyers, and K.R. Sloan, Jr. (in preparation) computer methods for reconstruction, display, and analysis of whole mounts: applications to human photoreceptor topography.
2. Drasdo, N. and C.W. Fowler (1974) Non-linear projection of the retinal maps in a wide-angle schematic eye. *Br. J. Ophthalmol.* 47: 609-613.
3. Farin, G. (1983) Smooth Interpolation to Scattered 3D Data. in *Surfaces in CAGD*, ed. Robert E. Barhill and Wolfgang Böhm, North-Holland Publishing Company, Amsterdam, pp. 43-63.
4. Frisen, L. (1970) The cartographic deformations of the visual field. *Ophthalmologica* 161:38-54.
5. Graham, R. L. (1972) An Efficient Algorithm For Determining The Convex Hull Of A Finite Planar Set. *Information Processing Letters* 1: pp. 132-133.
6. Østerberg, G.A. (1935) Topography of the layer of rods and cones in the human retina. *Acta Ophthalmol* 13 Suppl. 5 pp. 1-102.
7. Sibson, R. (1978) Locally Equiangular Triangulation. *The Computer Journal* 21:3 pp. 243-245.
8. Stone, J. (1981) *The whole mount handbook*. Sydney: Maitland.
9. Sloan, K.R. and C.M. Brown (1979) Color map techniques. *CG&IP* 10, 296-316.

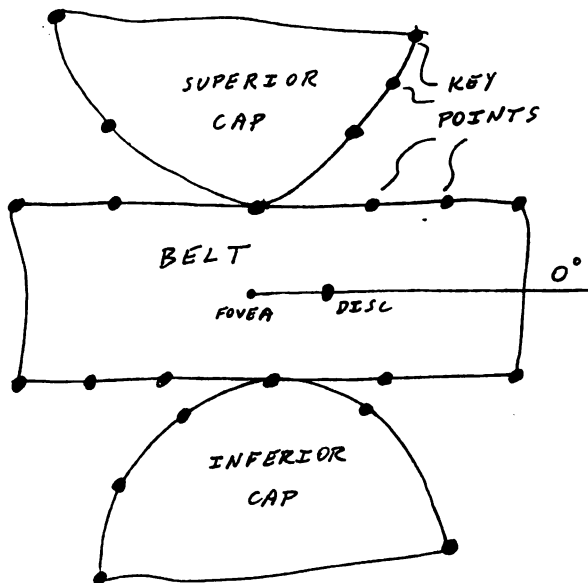


Figure 1. the three-piece dissection. The belt is roughly rectangular, and can be positioned on the sphere using the fovea and optic disc as landmarks. The inferior (superior) cap is positioned relative to the belt by means of key points which can be located in both the belt and the cap.



ing, and cute T_EX hacks, such as these silly pictures.



while skiing, he works as a Research Technologist at the University of Washington, where he is also studying Computer Science.



and knows what the pictures really mean.

Kenneth R. Sloan, Jr. was born in Millington, TN, on June 12, 1947. He received a Ph.D. in Computer and Information Science from the University of Pennsylvania. His research interests include graphics, (computer) vision, artificial intelligence, distributed computing,

David Meyers was born July 8, 1949. He attended the University of Colorado where he received his B.S. in Cell Biology. He came to Seattle in 1975. While not hanging from the end of a trapeze wire on a 505 class dinghy, or attempting to bury himself headfirst in a snowbank

Christine A. Curcio was born on November 16, 1950 and received a Ph.D. in Anatomy from the University of Rochester. She is interested in the organization of the retina, aging of the nervous system, and computer-assisted morphometry. She counted the rods and cones,

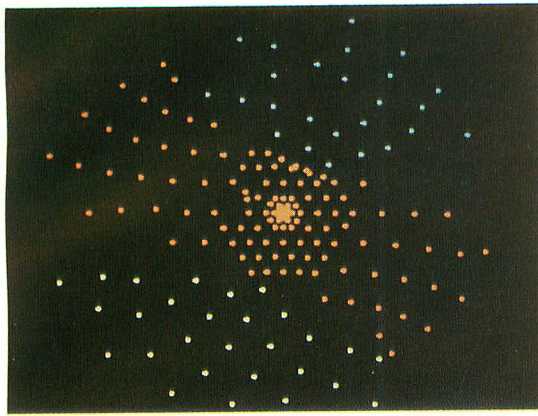


Figure 2. Sample points from a typical reconstruction, displayed using the polar azimuthal equidistant projection.

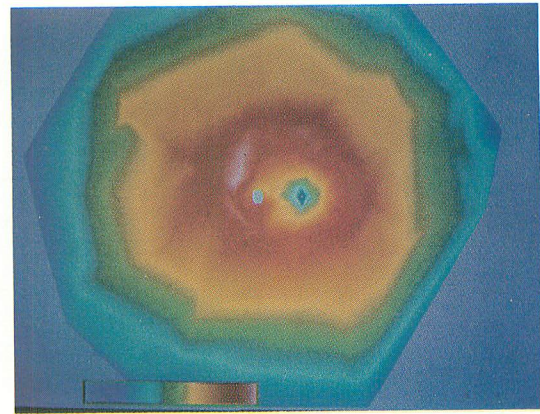


Figure 3. A display of Østerberg's rod density data, smooth shaded (in the plane) and false colored.

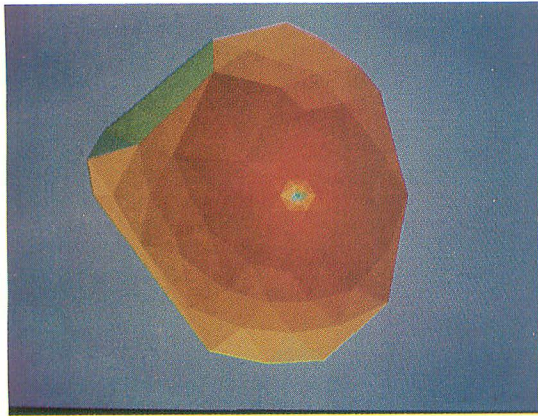


Figure 4. A rod map from a different eye, with the triangles shaded according to the average density at each vertex.

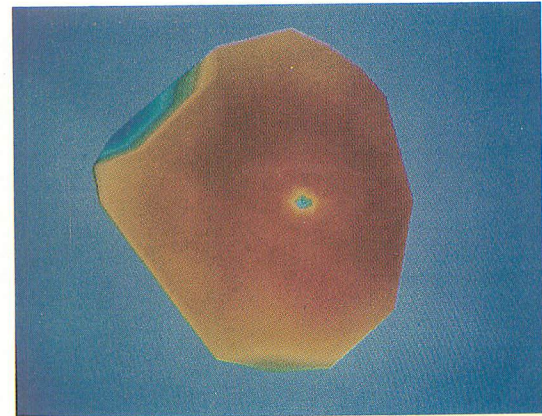


Figure 5. The same rod map as in Figure 4, displayed smooth shaded.

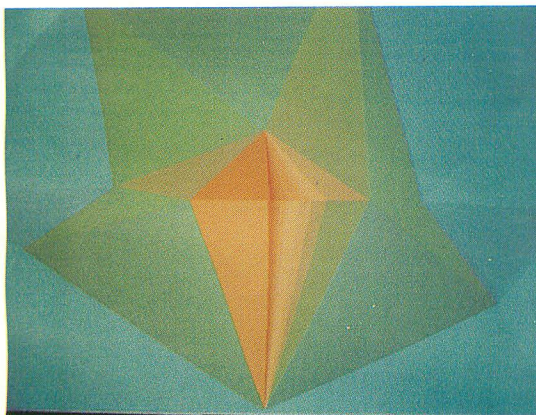


Figure 6. The central 8° of Østerberg's cone density data. Note the anisotropic sampling, and the "kite-shaped" pattern.

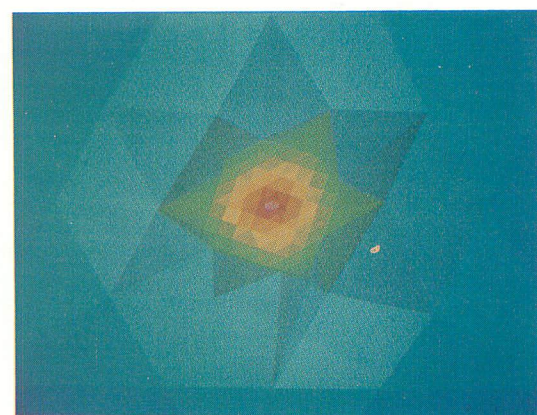


Figure 7. The central 8° of one of our recent eyes.

OPTICAL CHARACTER RECOGNITION OF TOUCHING CHARACTERS

S. Shlien and K. Kubota¹

Communications Research Centre
3701 Carling Avenue
Ottawa, Ontario, Canada
K2H 8S2

¹Visiting scientist from Nippon Telegraph and Telephone
Yokosuka Electrical Communication Laboratory,
Yokosuka-shi, 238 Japan

ABSTRACT

A method for isolating and recognizing characters in typeset text containing touching or overlapping characters is described. The method relies on vector quantization techniques to represent the text by an ordered sequence of codes. Characters or character fragments are extracted from this code using a form of string matching and network searches. Compatibility relationships are applied to these fragments to obtain a list of possible input strings. Our simulations on the Symbolics 3640 workstation do not indicate that this method would replace conventional template matching schemes.

KEYWORDS: optical character recognition, template matching, vector quantization, consistent labeling problem.

INTRODUCTION

With the continuous drop in the price of computer memory and VLSI components, it is easy to envision new and more powerful OCR devices becoming available. Such devices would utilize several different recognition strategies depending upon the quality of the input; they would be capable of learning a new font without human supervision; and they would contain large spelling dictionaries for applying contextual postprocessing [1-4]. Software development costs rather than the cost of hardware components would be the limiting factor to building such a system.

Automatic reading machines can process good quality typescript input at error rates approaching those of the average typist, however, only a few machines can handle typeset material containing proportionally-spaced characters [5]. A recurrent problem is the presence of touching characters which invariably results in segmentation and misclassification errors (eg. rn confused with m). Without gaps separating characters, the character extraction and classification process can no longer be performed independently, and this

increases the complexity of the recognition problem considerably.

There have been several studies [6-8] on the segmentation problem of touching characters. Some excellent results have been obtained using a combination of dynamic programming and template matching

[8]. A new and promising approach uses word shape matching [9]. Our interest in this problem has been motivated primarily by its similarity to the connected speech recognition problem.

In fluent spoken speech, there are no silence gaps between words [10]. The computational requirements for segmenting speech into words presently precludes the operation of any real time speech recognizer on a vocabulary of 5000 words [11].

In both speech recognition and optical character recognition, template matching techniques have been the preferred approach in commercial devices [12 and 13]. Though it is recognized that feature extraction techniques are more robust in handling multifonts, template matching techniques are fast and easy to implement and can handle poor quality input containing noise, voids and touching characters [13]. However, template matching schemes have little tolerance to minor type-face variations, of which there are more than 3000 in common usage [5]. Some of the newer schemes [14] attempt to apply combinations of these two techniques and hopefully reap the benefits of both methods.

Preprocessing techniques in speech recognition rely on data compression techniques (eg. linear predictive coding) in order to extract the essential information from the input data. Some of the more recent recognition methods also apply the rediscovered vector quantization coding schemes [15 and 16] to reduce the data to an input stream of symbols. The Hidden Markov Model [17-21]

is then used to interpret this stream of symbols. Though the accuracy of the vector-quantization based algorithms is lower than that of the conventional dynamic time warping techniques, the method requires an order of magnitude less processing time [20]. A further advantage of the Hidden Markov Model is that it avoids the need to explicitly code subjective rules about the interpretation of the patterns [22].

In our investigation, we exploited the vector quantization coder to extract the structural features of the characters. String matching techniques were then used to extract possible characters or character fragments from the text and network constraints were used to eliminate some of the interpretations.

Our study has so far been limited to the computer-generated fonts on a Symbolics Model/3640 workstation. This approach is effective for prototyping our design since it expedites the training process and factors out other processes in an OCR such as skew detection, correction for uneven illumination and distortions introduced by the optics system. Our simulations have provided us with useful information.

PREPROCESSING AND DATA REDUCTION

In the first step the image representation of a line of text is converted into an ordered list of discrete symbols to permit the application of one of many contextual pattern recognition schemes [1 and 23]. For example, the stream of symbols could be viewed as a grammar containing sufficient information to separate and identify the characters using syntactical pattern recognition schemes [24].

A short character string was displayed in a window of the Symbolics workstation using the TIMESROMAN12 font (one of a hundred fonts available in this workstation). Character spacing was adjusted so that adjacent characters either touched or overlapped in the horizontal direction. A low pass spatial gaussian filter [14] was applied to this window to simulate the optics of an OCR scanner and the resulting image was resampled to a lower resolution along a regular rectangular grid. The resampled image was thresholded to form a binary representation.

The sample column vectors extracted from this grid consist of sequences of ones and zeros which were then run length encoded. Using an appropriate distance measure and a training set of 5074 sample column vectors which were derived from

the characters in the TIMESROMAN12 font, the sample column vectors were grouped into 79 categories. The distance measure was based on the number of runs of black samples (ones), their position and their length. The categories were chosen so as to ensure that no particular training vector was further than a certain distance from the category template.

Using this library of category templates which we call the prototype list, the sample vectors in the text window were converted into a list of prototype codes which referenced the position of the closest matching prototype in the prototype list. This encoding scheme is similar to the vector quantization compression scheme applied to speech. The text data can be reconstructed from the code list with a nominal amount of distortion (Figure 1).

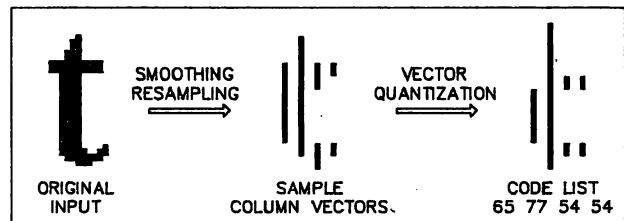


Figure 1. The characters in the TIMESROMAN12 font (39 points high including descenders) were smoothed and resampled with a unit variance two dimensional Gaussian filter and thresholded at a level of 0.3. The resampling density was chosen to yield 16 points per column. The sample column vectors were quantized to one of 79 producing a code-list as shown for the letter t.

SEGMENTATION AND CHARACTER RECOGNITION

Given the stream of prototype codes from a text window, the labeling of the codes as elements of characters is a form of the consistent labeling problem [25]. Each column vector may be a part of many possible characters however local contextual constraints limit the solution. The consistent labeling problem is an NP-problem and includes the graph homomorphism problem, the graph colouring problem, the packing problem, the scene labeling problem, the shape matching problem, the constraint satisfaction problem and theorem proving [25]. For this reason, the Symbolics work station was considered as a suitable tool for performing this study.

A related problem is the restoration of spaces between words in the following text string

'heworkswithcomputers'.
Using a large spelling dictionary, the computer can find nine possible words

embedded in the string:

computer he or with works
computers it put work.

However, assuming that every letter must belong to exactly one word in the dictionary, the computer will arrive at the correct solution

'he works with computers'.

The dictionary search is the most computationally intensive component of this problem. Special string comparison VLSIs [26] allow rapid solution of this problem on a PC with a large spelling dictionary.

In the OCR problem, two additional levels of complexity are introduced. Since the position of the scanning grid varies randomly with respect to any character [27], the same character may have many possible prototype code sequences (Figure 2). Other factors increasing this variability are the evenness of the lighting and the ink absorbing characteristics of the paper. Secondly, when adjacent characters are in contact, the initial or final sequence of codes of a character may be distorted or lost. The OCR must rely on the information in the central portion of the character. For some of the narrow characters (such as l, i, r, t and the punctuation marks), this poses a major difficulty.

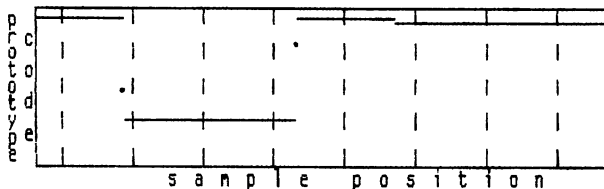


Figure 2. The prototype code (a number between 1 and 79) versus horizontal sampling position is shown for the upper case character L. The vertical dashed lines indicate sampling positions for a sampling rate equal to the vertical sampling rate. Note some prototype codes represented by the two dots in the plot have a very narrow range of existence. Such codes are easily missed during the training process.

To allow for this variability, the string matching procedure was implemented as a network search. A cross reference table was associated with every prototype code. This table lists all the characters that reference this code and the relative position in the character. In the previous example, the cross reference table associated with the letter o would include the following possible interpretations:

(or 1) (work 2) (works 2) (computer 2)
(computer 2) (program 3) ...

The number associated with each word

indicates the position of the character o in the word.

Given an unknown string, the algorithm attempts to create an interpretation list which satisfies the following constraint where c_i represents the letter at column i and (w_i, s_i) is one of the interpretations corresponding to c_i . Two interpretations (w_i, s_i) and (w_{i+1}, s_{i+1}) corresponding to c_i and c_{i+1} are compatible if one of the following conditions are satisfied.

1 $w_{i+1} = w_i$ and $s_{i+1} = s_i$
or

2 $w_{i+1} = w_i$ and s_i is the last character position w_i and s_{i+1} is the first character position of w_{i+1} .

In our OCR application the two conditions were relaxed to allow the additional variability.

IMPLEMENTATION AND RESULTS

A cross reference table was created by resampling and vector quantizing the characters at a high sampling rate. In order to limit the table to a workable size, the positional information, s_i , was stored at lower resolution. On average, there were 27 interpretations associated with each of the 79 prototype codes.

The recognition software was divided into two sections. In the first pass, character fragments were extracted from the code list by searching for the longer chains of compatible interpretations. In the next pass, the algorithm attempted to patch the character fragments together in order to obtain a consistent interpretation. Frequently, the algorithm would return several consistent interpretations; they were ranked by the average character visibility in the text window.

The algorithm was implemented in compiled LISP on our Symbolics 3640 workstation with 512K words memory and without a floating point accelerator. A string of 7 touching characters was processed in approximately one minute. Eighty percent of this time was spent preprocessing the data and vector quantizing the sampled column vectors. The use of a VLSI pattern matching chip for performing the vector quantization [28] could reduce this time.

The algorithm had almost no difficulty identifying the isolated characters. However, it was necessary to use the maximum horizontal resolution (double the vertical resolution). The algorithm did not always distinguish some of the punctuation marks (;) and confused the T

THAT	THAT THAT THAT THAT	that	that thar
WITH	WITH WITH	with	with wirh
THIS	THIS THIS	this	this
FROM	FROM PROM FRCM PRCM	from	from from fmm frtxn
HAVE	HAVE	have	have
THEY	THE' THE' THB' THB'	they	they
WHICH	WHICH	which	which whioh whinh
WERE	WERE WBRE	were	were wete wert; wen;
THERE	THERE THERE THBRE THBRE	there	there thete
WHEN	WHEN WHBN	when	when
WILL	WILL WILL	will	will
MORE	MORE MORE	more	more mote mcre more
SAID	SAID SAID	said	said saill saill
WHAT	WHAT WHAT	what	what
ABOUT	ABOUT ABOUT ABOUT ABOUT	about	about about about
ONLY	ONLY ONLY	only	only only
OTHER	OTHER OTHER OTHER OTHER	other	other other crher
SOME	SOME SCME SOME SCME	some	some some some sotne

Figure 3. Results of applying the recognition algorithm on some common 4 and 5 letter English words displayed in TIMESROMAN12 font in both upper case and lower case characters. The second and fourth columns contain the list of possible character strings which match the input in decreasing order of average character visibility.

with the three character sequence 'I'.

The algorithm was next tested on a set of the twenty most common 4 and 5 letter words which were displayed in both upper and lower case touching characters. The algorithm usually returned several consistent interpretations (eg. more, mote, mcre, rmore), however the first choice was almost always the correct choice (Fig. 3). Occasionally, the algorithm failed to recognize a particular letter (eg. Y in THEY). The failure was traced to a missing entry in the cross reference table which resulted in a broken chain of interpretations for this letter.

The next test consisted in applying the OCR to the TIMESROMAN15 font. Though the

resampling process automatically normalizes the character dimensions, the smoothing filter was fixed and resulted in some small differences for the larger font. The OCR initially failed to recognize the new characters in this font, but when the cross reference table was extended to include these characters the algorithm was able to handle both fonts.

DISCUSSION OF RESULTS

The purpose of this study was to examine the feasibility of using string matching techniques to isolate and recognize proportionally-spaced characters which touched or overlapped. We represented the unknown string of characters by a

sequence of equally spaced sample column vectors which were extracted from the text. The column vectors were categorized into about eighty types and their codes were sent to the classifier which applied string matching techniques to extract and identify the characters.

Our simulations so far were restricted to a computer generated font, TIMESROMAN12 on a Symbolics workstation. The algorithm succeeded in identifying most of the characters but encountered difficulties distinguishing some pairs such as (r,t), (o,c), (l,l) and (T,I) when the beginning or end of the characters were obscured by neighbouring characters.

Our preliminary investigations do not demonstrate any clear advantage of the string matching approach over the conventional template matching schemes (eg. [8]). Despite the use of a cross reference table to accelerate the matching process, the recognition process was still complex. A sequence of filtering processes were required to remove many of the character fragments which tended to impede the recognizer.

The accuracy measurements indicate that a higher vertical sampling rate and a larger prototype list would be needed to preserve the details of the characters. Furthermore, in order to handle italics and Greek mathematical symbols, the prototype list would need to double in size. Increasing the prototype list would increase the cross reference table, the network search time and the sensitivity of the recognizer to any image noise.

REFERENCES

1. Godfried T. Toussaint, The use of context in pattern recognition, Pattern Recognition, vol. 10, 189-204, 1978.
2. Ching Y. Suen, n-Gram statistics for natural language understanding and text processing, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2, 164-172, 1979.
3. A.R. Hanson, E.M. Riseman and E. Fisher, Context in word recognition, Pattern Recognition, vol. 8, 35-44, 1976.
4. Rajjan Shinghal and Godfried T. Toussaint, Experiments in text recognition with the modified Viterbi algorithm; IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2, 184-193, 1979.
5. George Nagy, Optical character recognition - theory and practice, Handbook of Statistics, (ed. Krishnaiah and Kanai), North Holland Publishing, 621-649, 1982.
6. R.L. Hoffman and J.W. McCullough, Segmentation methods for recognition of machine-printed characters, IBM Journal of Research and Development, vol. 15, 153-165, 1971.
7. V.A. Kovalevsky, Image Pattern Recognition, Springer-Verlag, chapter VIII, 1980.
8. R.G. Casey and G. Nagy, Recursive segmentation and classification of composite character patterns, 6 th International Conference on Pattern Recognition, 1023-1026, 1982.
9. Jonathan J. Hull, Word shape analysis in a knowledge-based system for reading text, 2 nd Conference on Artificial Intelligence Applications, 114-119, 1985.
10. Ronald A. Cole (ed.) Perception and Production of Fluent Speech, Lawrence Erlbaum Associates, 1980.
11. Frederick Jelinek, The development of an experimental discrete dictation recognizer, Proc. of IEEE, vol. 73, 1616-1624, 1985.
12. Victor W. Zue, The use of speech knowledge in automatic speech recognition, Proc. of IEEE, vol. 73, 1602-1614, 1985.
13. Arthur W. Holt, The impact of new hardware on OCR designs, Pattern Recognition, vol. 8, 99-105, 1976.
14. A. Lashas, R. Shurna, A Verikas and A. Dosinas, Optical character recognition based on analog preprocessing and automatic feature extraction, Computer Vision, Graphics and Image Processing, vol. 32, 191-207, 1985.
15. R.M. Gray, Vector Quantization, IEEE Acoustics, Speech and Signal Processing Magazine, vol. 1, 4-29, 1984.
16. John Makhoul, Salim Roucos and Herbert Gish, Vector quantization in speech coding, Proc. of IEEE, vol. 73, 1551-1588, 1985.
17. J.K. Baker, The Dragon System - an overview, IEEE Trans. on Acoustics, Speech and Signal Processing, vol.

ASSP-23, 24-29, 1975.

18. L.E. Baum, T. Petrie, G. Soules and N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic function of Markov chains, Annals of Mathematical Statistics, vol. 41, 164-171, 1970.
19. S.E. Levinson, Structural methods in automatic speech recogniton, Proceedings of the IEEE, vol. 73, 1625-1649, 1985.
20. L.R. Rabiner, S.E. Levinson and M.M. Sondhi, On the application of vector quantization and Hidden Markov Models to speaker-independent, isolated word recognition, The Bell System Technical Journal, vol. 63, 627-642, 1984.
21. Frederick Jelinek, Continuous speech recognition by statistical methods, Proc. of IEEE, vol. 64, 532-556, 1976.
22. R. Nag, R.H. Wong and F. Fallside, Script recognition using Hidden Markov Models, International Conference on Acoustics, Speech and Signal Processing, 1986.
23. Robert M. Haralick, Decision making in context, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-5, 417-428, 1983.
24. K.S. Fu, Syntactical Pattern Recognition and Applications, Prentice-Hall, Englewood Cliffs, N.J., 1982.
25. Robert M. Haralick and Linda G. Shapiro, The consistent labeling problem: part I, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. PAMI-1, 173-184, 1979.
26. Steve Rosenthal, The PF474, BYTE, vol. 9, no. 12, November 1984.
27. W.R. Throssell and P.R. Fryer, The measurement of print quality for optical character recognition systems, Pattern Recognition, vol. 6, 141-147, 1974.
28. Grant Davidson, Terry Stanhope, R. Aravind and Allen Gersho, Real-time speech compression with a VLSI vector quantization processor, Proc. of International Conference on Acoustics, Speech and Signal Processing, 1985.

CONTOUR LINE REGION SEGMENTATION

(Extended Abstract)

Lawrence O'Gorman

Gerald I. Weil

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

In images such as of contour maps, fingerprints, and electric fields, regions of contour lines can be distinguished, and these regions are often used for image understanding. In this work, such images are collectively termed contour line images. The objectives are to determine the properties by which contour line regions are characterized, and to develop an approach using these properties to automatically determine regions. An algorithm is proposed to group lines into regions. This is based in part on the parallel-adjacency criterion which is defined here. The algorithm has been applied to several contour line images, and the resultant regions are shown.

KEYWORDS: contour line regions, image segmentation, pattern recognition

1. Introduction

It is a simple matter for a person to locate a knot in wood grain, or to recognize a pattern in marble. Patterns on contour maps can be recognized as locations of rivers or steep grades. Other constant-value plots such as flux line fields in electromagnetics and isobar maps in meteorology are interpreted by the *pattern* of lines (rather than by individual lines) to understand global information in the image. Differences in the shape of contour patterns in fingerprint images can be used, even by non-experts, to distinguish different prints. All these examples involve images of contour lines in which regions of line patterns are discerned by human viewers, and from which interpretation or recognition is performed. Although human recognition of these patterns seems trivial, it is no trivial task for a computer. In this paper, we examine properties which characterize regions in contour line images, and propose an approach for performing automatic determination of these regions.

We wish to deal with contour line images, irrespective of what the contour lines represent. The

expression *contour line image* will refer to an image made up of lines (where lines can be curved) which run approximately parallel to adjacent lines, at least over short distances. The spacing between adjacent lines is often small, and the lines may be densely packed. Therefore, groups of lines appear as regions rather than boundaries. Examine Figure 1, and determine the number of contour line regions in each diagram based on the "similarity of pattern" within each region. Our experience shows that there is a general consensus in the number of regions perceived for each. It is this response that we desire to emulate by machine. (Our informal examination of the human perception of these regions is inadequate to claim that this is a universal response, however this does not lessen the interest in proposing a method for emulating what we have found to be a common response.) The common responses given for the number of regions in each diagram of Figure 1 are shown in Figure 2.

2. Method

Our objectives are to determine the properties which characterize contour line regions, and to describe an approach which utilizes these properties in order to segment regions. It is necessary to clarify some terminology before describing the properties and approach. A *contour line*, or *line*, is a straight or curved sequence of contiguous points between two endpoints. A *segment* is a straight line fit to a portion of a line. We will refer to *regions* of lines, and *groups* of segments.

First, we combine descriptions of some of the properties of contour line images into the expression **parallel-adjacency**. The parallel-adjacency criterion specifies that if a pair of segments are:

1. adjacent (i.e. not separated by other lines),
2. close in distance,
3. approximately parallel, and
4. overlap by a specified amount,

then the lines to which the segments belong are potentially in the same region. In the algorithm to be described below, pairs of segments are first compared for parallel-adjacency, and groups of line segments are built by these pairwise comparisons. Then, in the same manner, adjacent lines are compared pairwise to determine if they are similarly comprised of segments of the same parallel-adjacency groups. In this way, lines are combined into *contour line regions*.

The main steps of the algorithm are listed below:

1. Split lines at all junctions (bifurcations and crosses in lines).
2. Perform piecewise straight-line fitting so that each line is comprised of straight line segments.
3. Construct an adjacency list of the straight-line segments. This segment adjacency list (SAL) contains, for each segment, all other segments meeting criteria for distance proximity, approximate parallelism, and non-zero overlap with respect to that segment.
4. Merge the segments of the SAL into groups on the basis of pairwise similarity of line segments due to the parallel-adjacency criterion. The result is the segment group list (SGL).
5. Consider each line in its entirety (made up of the straight line segments), and group the lines, again in pairwise fashion, based on line adjacency and similar composition of line segments from the SGL. The result is the line region list (LRL) containing line composition of each contour line region.

3. Results

The algorithm was applied to each of the diagrams in Figure 1. The results are shown in Figure 2. For these synthesized images, which contain no noise, the regions found by the algorithm are consistent with our expectation and approach. Figure 3 shows the results of the algorithm as applied to a thinned fingerprint image which contains broken, short, and isolated lines. The 3 largest regions (in terms of number of lines per region) which are found by the algorithm are shown. We are currently working to better establish the relationships of the algorithm parameters to the image characteristics — especially for images containing noise.

4. Summary

The objectives addressed in this work are to determine properties which characterize contour line regions, and to automatically distinguish those regions. In the context of this work, contour line images consist of a large number of lines, where adjacent lines are closely spaced, overlap, and are approximately parallel. From these properties, the parallel-adjacency criterion is defined and used to associate piecewise segments of different lines into groups. Contour line regions are then found by pairwise merging of lines which are similarly comprised of groups of segments. Experiments have shown that the regions determined by the algorithm as applied to both synthetic and real images are consistent for our approach.

A short description of the method has been given in this extended abstract. Continuing work will give a better understanding of the performance of the algorithm for different conditions and types of images, and a more detailed description will be given in a future paper.

Relevant Literature

- K.A. Stevens, "The visual interpretation of surface contours.", *Artificial Intelligence* 17, 1981, pp47-74.
- G.J. Agin, "Representation and description of curved objects." (Ph.D. dissertation), AIM-173, Stanford AI Lab, October, 1972.
- O. Nakamura, K. Goto, T. Minami, "Fingerprint classification by directional distribution patterns", *Systems, Computers, Controls*, Vol. 13, No. 5, 1982, pp. 81-89.
- M. Kawagoe, A. Tojo, "Fingerprint pattern classification.", *Pattern Recognition*, Vol. 17, No. 3, 1984, pp. 295-303.

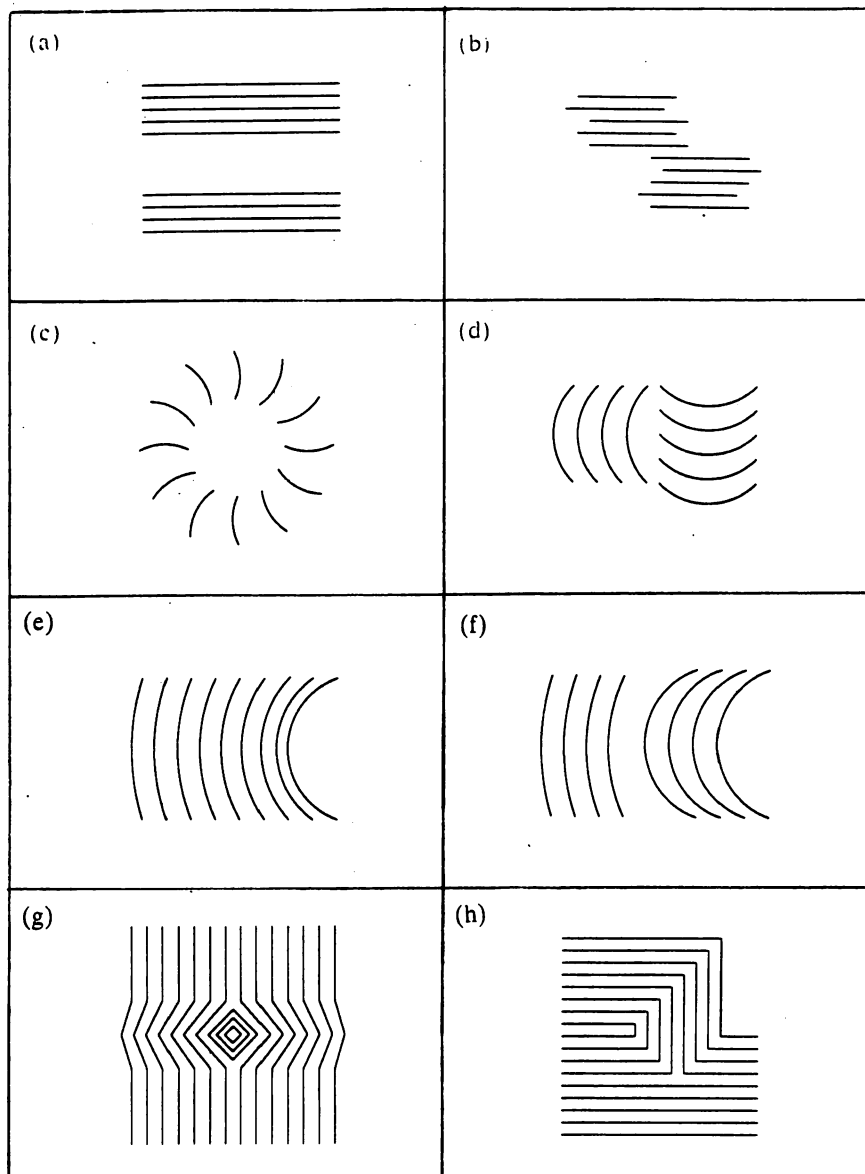


Figure 1. How many contour line regions are there in each image?

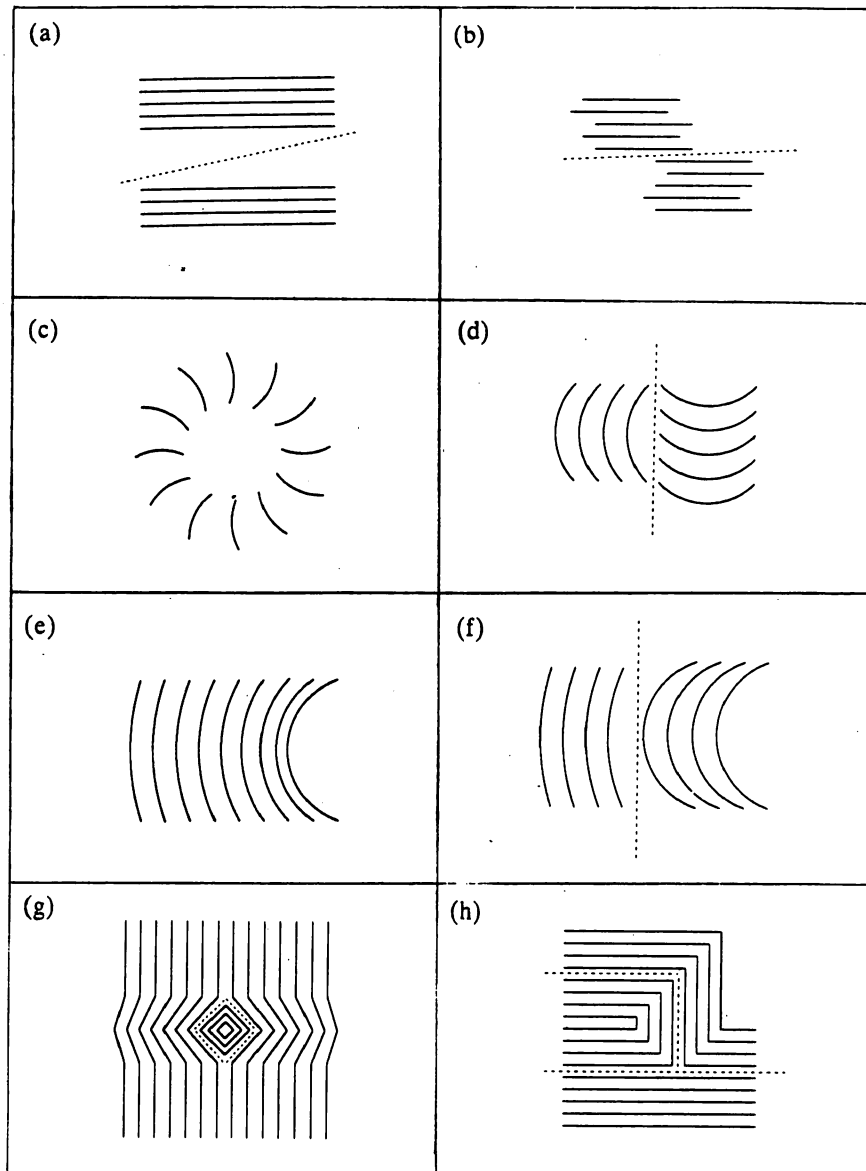


Figure 2. Contour line regions found by the algorithm for the images in Figure 1. Number of regions: (a) 2. (b) 2. (c) 1. (d) 2. (e) 1. (f) 2. (g) 2. (h) 3.

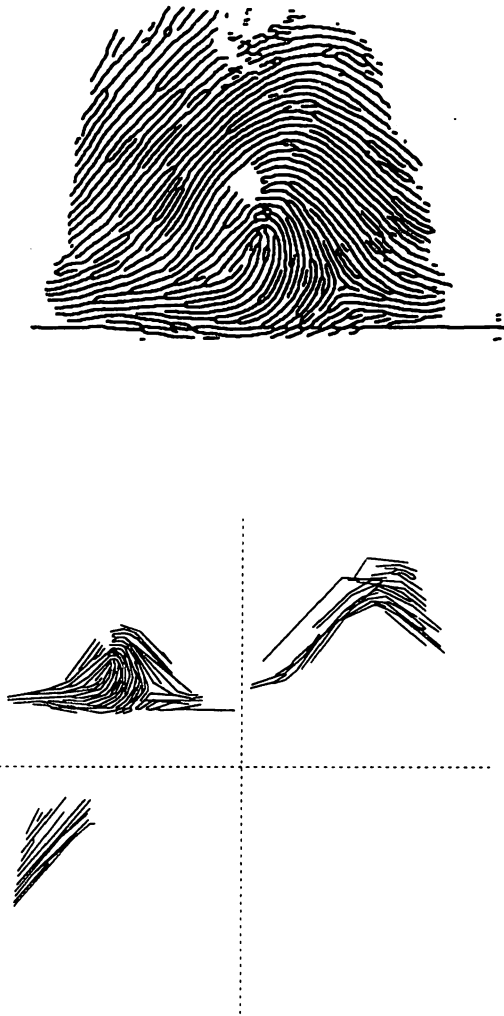


Figure 3. Fingerprint image, and the regions found.

AUTHOR INDEX / RÉPERTOIRE DES AUTEURS

Adjouadi, M.	307	Kenk, E.	279
Archibald, C.C.	293	Kubota, K.	390
Armstrong, W.W.	147	Kurz, R.	380
Badler, N.I.	115	Lake, R.	147
Barr, J.M.	331	Lefevre-Fonollosa, M.-J.	284
Barsky, B.A.	241	Levine, M.D.	260, 380
Booth, K.S.	82, 91, 194	Lewis, J.P.	173
Bouthemy, P.	350	Lewis, J.W.	32
Brault, J.-J.	375	Liakopoulos, A.	366
Brzakovic, D.	366	Liang, P.	313
Cachola, D.G.	152	MacKay, S.A.	98
Caelli, T.M.	343, 356	Magnenat-Thalmann, N.	213
Calvert, T.W.	121, 300	Malowany, A.S.	380
Carroll, J.M.	186	Mansouri, A.-R.	380
Coggins, J.M.	229	Meyers, D.	385
Cruchant, H.	284	Myers, B.A.	62
Curcio, C.A.	385	Nagendran, S.	356
Davis, W.A.	235, 287	Nemoto, K.	43
DeRose, T.D.	241	Nguyen, P.T.	267
Drewery, K.	131	Nichols, M.	26
ElMaraghy, H.A.	15	Nisselson, J.	1
Fay, F.S.	229	Olsen, Jr., D.R.	66
Ferch, H.J.	11	Omachi, T.	43
Fogarty, K.E.	229	Oppenheimer, P.	254
Forest, L.	213	Ostrovsky, R.	20
Forsey, D.R.	194	O'Gorman, L.	396
Fournier, A.	49, 164	Paeth, A.W.	77, 91, 194
Franklin, W.R.	26	Peachey, D.R.	37
Fuchs, H.	193	Pearce, A.	136, 217
Gardner, B.R.	20	Pellicano, P.N.E.	370
Glass, G.J.	180	Pentland, A.P.	223
Goldberg, M.	273	Plamondon, R.	375
Goodenough, D.G.	266, 273	Plunkett, G.W.	273
Grant, E.	104	Pointing, T.	188
Green, Marc	337	Prusinkiewicz, P.	158, 247
Green, Mark	71, 147	Ramnaud, D.	213
Greene, N.	108	Ridsdale, G.	121
Grindal, D.A.	164	Samaddar, S.	26
Gross, J.R.	241	Schlag, J.F.	202
Hanrahan, P.	56	Schoeler, P.	49
Heckbert, P.S.	207	Schrack, G.F.	152
Hewitt, S.	121	Shlien, S.	390
Higgins, T.M.	82	Sims, P.	361
Hill, D.	136	Singh, G.	71
Holynski, M.	20	Sloan, Jr., K.R.	385
Hong, W.	260	Sondheim, M.	279
Hoskins, J.A.	7	Sternberg, S.R.	293
Hoskins, W.D.	7	Streibel, D.	158
Hutber, D.	361	Tanimoto, S.L.	349
Hwang, C.H.	287	Tanner, P.P.	98
Jernigan, M.E.	325	Thalmann, D.	213
Kamal, M.R.	380	Thornton, R.W.	180

Todhunter, J.S.	313	Whitted, T.	104
Tsotsos, J.	131	Wilhelms, J.	141
Tuori, M.	188	Wu, P.	26
Turpin, D.	279	Wyvill, B.	136, 217
Walford, A.E.J.	325	Wyvill, G.	136, 217
Walters, D.	318	Xie, S.	300
Wang, X.	235	Yee, B.	279
Weil, G.I.	396		